
ELSEVIER Applied Mathematics and Computation 92 (1998) 153–193

APPLIED MATHEMATICS AND COMPUTATION

Numerical approaches for solution of differential equations on manifolds

R. Sudarsan [a,*], S. Sathiya Keerthi [b]

[a] *Department of Engineering, George Washington University, Washington, DC 20052, USA*
[b] *Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India*

Abstract

Numerical approaches for the solution of vector fields (differential equations defined on a manifold) have attracted wide attention over the past few years. This paper first reviews the various numerical approaches available in the literature for the solution of vector fields namely, Parameterization approach, Constraint Stabilization approach, and Perturbation approach (PA). In the process, the paper also makes the following useful contributions: an expanded analysis and a new perturbation scheme for the PA; and a new way of choosing integration error tolerances for the parameterization approach. A comparison of all the approaches is carried out, both by means of a crude cost analysis as well as by studying their numerical performance on examples of vector fields arising from constrained mechanical systems (CMS). Based on this comparison, recommendations are made for a proper choice of a suitable approach. Overall, the PA performs 'better' than the other approaches.

AMS classification: 65L05; 65L06; 70H35

Keywords: Vector fields; Differential-algebraic equations; Manifolds; Local parameterization; Constraint stabilization; Euler–Lagrange equations; Multibody systems; Numerical ODEs

* Corresponding author. E-mail: sudarsan@cme.nist.gov.

0096-3003/98/$19.00
PII: S0096-3003(97)10065-0

1. Introduction

Numerical approaches for the solution of vector fields (differential equations defined on a manifold) have attracted wide attention over the past few years. This paper first reviews the various numerical approaches available in the literature for the solution of vector fields namely, Parameterization approach, Constraint Stabilization approach, and Perturbation approach (PA). In the process, the paper also makes the following useful contributions: an expanded analysis and a new perturbation scheme for the PA; and a new way of choosing integration error tolerances for the parameterization approach.

The paper is organized as follows. In Section 2, we define vector fields and discuss some applications. The various existing approaches for the solution of vector fields are discussed in Section 3. In Section 4, a comparison of all the approaches is carried out, both by means of a crude cost analysis as well as by studying their numerical performance on examples of vector fields arising from constrained mechanical systems (CMS). In the final section based on this comparison, recommendations are made for a proper choice of a suitable approach. Overall, the PA performs *better* than the other approaches.

2. Vector fields

We begin by formally defining the vector field that is to be solved numerically. Let $\mathcal{M}$ be an $(n-m)$ dimensional manifold in $\mathbb{R}^n$ defined by

$$g(x) = 0, \tag{2.1}$$

where $g: \mathbb{R}^n \to \mathbb{R}^m$ is a smooth function. Assume that $\exists$ an open set $\mathcal{O}$ in $\mathbb{R}^n$ containing $\mathcal{M}$ such that $g_x(x)$, the $m \times n$ Jacobian of g at x, satisfies $\text{rank}(g_x(x)) = m \ \forall x \in \mathcal{O}$. Let

$$\dot{x} \triangleq \frac{\mathrm{d}x}{\mathrm{d}t} = f(x), \quad t \in [t_0, t_f] \tag{2.2}$$

define a vector field on $\mathcal{M}$. In other words, if $T_x\mathcal{M}$ denotes the tangent space of $\mathcal{M}$ at $x \in \mathcal{M}$, then $f(x) \in T_x\mathcal{M} \ \forall x \in \mathcal{M}$. Usually a smooth extension of f to an open set in $\mathbb{R}^n$ containing $\mathcal{M}$ is available. We will assume this to be the case for, when dealing with numerical methods points slightly off from $\mathcal{M}$ are obtained and there may be a need to evaluate f there.

Remark 2.1. Our assumption that g and f are time invariant, i.e., they do not explicitly depend on t, is only for the purpose of simplifying the notations and some of the discussions and most of the ideas can be extended to time varying case. The case of time-varying g and f is discussed in detail in Refs. [1,2].

2.1. Numerical solution of vector fields

Let us assume that the vector field (2.2), is solvable, i.e., given any $x_0 \in \mathcal{M}$ there exists a unique solution, $x: [t_0, t_f] \to \mathcal{M}$ that satisfies $x(t_0) = x_0$ and (2.2). See Ref. [3] for some theory on solvability. Our aim is to obtain a numerical approximation of $x(\cdot)$. Since the facts, $x_0 \in \mathcal{M}$ and $x(\cdot)$ satisfies (2.2), automatically guarantee $g(x(t)) = 0 \ \forall t \in [t_0, t_f]$, we could simply ignore (2.1) and numerically solve (2.2) using a well-known integration [4–8] method for ordinary differential equations (ODEs) to obtain an approximate solution, $\tilde{x}(\cdot)$. There are two strong objections to such an approach which excludes (2.1) from the solution process: (i) $\tilde{x}$ may violate (2.1) badly; and (ii) the information that (2.1) is an invariant along the solution can be used to check, and perhaps improve, the accuracy of the solution i.e., any available additional information when used properly can set a better solution in terms of accuracy. Let us go in to these issues in more detail.

Suppose (2.2) is integrated using global error control, i.e., given a tolerance τ, $\tilde{x}$ satisfies

$$\|\tilde{x}(t) - x(t)\| \leqslant \tau \quad \forall t \in [t_0, t_f], \tag{2.3}$$

where $\|\cdot\|$ is an appropriate norm of $\mathbb{R}^n$ used in integration. Take g^i, the ith component of g. Fixing t, the first-order truncated Taylor series of g^i expanded around $x(t)$ and evaluated at $\tilde{x}(t)$ is

$$g^i(\tilde{x}(t)) = g^i(x(t)) + g^i_x(\eta_i)(\tilde{x}(t) - x(t)) = g^i_x(\eta_i)(\tilde{x}(t) - x(t)), \tag{2.4}$$

where η_i is some point on the line segment joining $\tilde{x}(t)$ and $x(t)$ and g^i_x is the ith row of g_x. Thus

$$|g^i(\tilde{x}(t))| \leqslant \|g^i_x(\eta_i)\| \ \|\tilde{x}(t) - x(t)\| \leqslant \|g^i_x(\eta_i)\|\tau. \tag{2.5}$$

Therefore, if for each i, g^i_x is a bounded function with a reasonable bound on its norm, then the violation of (2.1) will not be severe.

Unfortunately, the current state of research in global error control is weak [5]. The paper by Skeel [9] explains several ways of handling global errors in ODEs. Also, methods which try controlling the global error are very expensive. All popular integration codes only employ local error control where integration is done via a number of successive time steps, the computations in one time step being done as if integration in the previous time steps was done exactly. When local error control is used, the violation of (2.1) can build up and become huge for large t. Since (2.1) is a fundamental constraint in our problem, a huge violation of it is certainly objectionable. The following example illustrates a case of severe violation in (2.1)

Example 2.1. Let $x = (x^1, x^2)$. Consider the oscillator equations,

$$\dot{x}^1 = x^2, \quad \dot{x}^2 = -x^1 \tag{2.6}$$

defined on the manifold $\mathcal{M}$ whose describing equation is $g(x) = (x^1)^2 + (x^2)^2 - 1 = 0$. Suppose $x_0 = (1, 0)$ and that the differential equations are solved by a first-order Taylor series method with local error control, the local error estimate being provided by the second-order derivative of x. Let τ denote the relative tolerance used in doing the integration.

More precisely, the details of going from x_k, the solution approximant at $t = t_k$ to x_{k+1} at $t_{k+1} = t_k + h$ are as follows. Given h, the first-order Taylor series formula gives $x_{k+1} = x_k + hf_k$, where $f_k = (x_k^2, -x_k^1)$. Suppose $v(t_k + h)$ denotes the true local solution, i.e., the solution of (2.6) with the initial condition $v(t_k) = x_k$. Define $e(h)$, the relative local error tolerance as $e(h) = \|x_{k+1} - v(t_k + h)\|/\|x_k\|$, where $\|x\|^2 = (x^1)^2 + (x^2)^2$. An estimate of $e(h)$ is $h^2/2$. This is obtained from the second-order Taylor series approximation, $v(t_k + h) \approx x_k + hf_k + (h^2/2)s_k$, where $s_k = (-x_k^1, -x_k^2)$. Since the aim is to keep $e(h) \leqslant \tau$, the optimum step size based on the error control is $h = \sqrt{2\tau}$. This is, roughly, the way most numerical methods do local error control.

Suppose integration is done until one revolution is complete, i.e., until x^1 changes from a negative to a positive value. A closed form expression for $g(x_k)$ can be obtained as follows. Observing that $\|x_{k+1}\|^2/\|x_k\|^2 = (1 + h^2) = (1 + 2\tau)$, we get $g(x_k) = (1 + 2\tau)^k - 1$. The number of integration steps needed to complete a revolution, N, can be obtained as $\lceil 2\pi/\theta \rceil$, where: $\theta = \tan^{-1}\sqrt{2\tau}$ is the angle made by x_k and x_{k+1} at the origin; and, for $\mu \in \mathbb{R}$, $\lceil \mu \rceil$ = the smallest integer which is greater than or equal to μ. Table 1, which gives $g(x_N)$ for various τ values, clearly points to an alarming violation of $g(x) = 0$.

In this example the cause of growth in the size of g is basically due to the inadequacy of local error control to maintain the global error within reasonable values. There is an unstable growth in the global error which is contributed by both, the problem solved and the method used. Although, typically, the growth of global error is not as bad as in this example, there is always an uncertainty in the effect of local errors on the global error. Therefore, when local error control is employed to solve (2.2), it is important to enforce (2.1) in the solution process. It will be ideal to require the numerical approximation, $\tilde{x}(\cdot)$ to satisfy (2.1) for all t. If that is difficult, it will even be sufficient if $\tilde{x}(\cdot)$ satisfies (2.1) at the end points of each integration time step because, an analysis similar to (2.3)–(2.5) that uses the local solution $v(\cdot)$ instead of $x(\cdot)$, can then be used to obtain reasonable bounds on $\|g(\tilde{x}(t))\|$ at other values of t.

Table 1
Constraint violation for various tolerances

τ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
$g(x_N)$	14.4	1.437	0.3254	0.093	0.0285	0.0089

In Example 2.1 the growth in the violation of (2.1) has provided a valuable means of detecting the deterioration in solution accuracy. (Note, however, that (2.1) is only a partial check because satisfying (2.1) well does not mean that the error in the solution is small.) Furthermore, (2.1) can be useful in other ways. Appropriate corrections on the numerical solution of (2.2), incorporated to force (2.1), may lead to a more accurate solution of (2.2). This is our belief and we will give supporting arguments to this in Section 3.4, where we discuss the PA. For instance, in Example 2.1 suppose we do the following alteration (manifold correction) after each integration step: compute $p = \arg\min\{\|x - x_{k+1}\|: g(x) = 0\}$, the point on $\mathcal{M}$ closest to x_{k+1}, and reset $x_{k+1} := p$. Then we get a much more accurate solution than before. Such a correction, in fact, forms the basis of the PA.

2.2. Applications

The problem of numerically solving (2.1), (2.2) arises in a number of applications such as CMS, flexible multibody systems [10,11] simulation of control systems modeled by DAEs, semiconductor device simulation [12], numerical curve tracing [13], homotopy curve tracing, handling physical invariants, and constrained optimization. Here we highlight a few important ones.

(a) CMS: Euler–Lagrange equations: The Euler–Lagrange equation that describes the motion of CMS can be written in the form:

$$M(q)\ddot{q} + J^{t}(q)\lambda = Q(\dot{q}, q), \tag{2.7}$$

$$0 = \phi(q), \tag{2.8}$$

where $q \in \mathbb{R}^n$ is the vector of generalized coordinates, $M(q) \in \mathbb{R}^{n\times n}$ is the generalized mass matrix, $\phi: \mathbb{R}^n \to \mathbb{R}^m$ is a nonlinear mapping that defines the constraints (kinematical), $J = \partial\phi/\partial q$ is the Jacobian of ϕ with respect to q (J^t denotes the transpose J), $\lambda \in \mathbb{R}^m$ is the vector of Lagrange multipliers associated with the constraints, and $Q: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ is a mapping that defines generalized external forces. Here time invariance of M, ϕ, Q, and J is assumed only to simplify the notations. The system of equations (2.7), (2.8) is an index-3 DAE. The index can be lowered by a procedure called index reduction. The index-1 form of the Euler–Lagrange equations can be written as,

$$\begin{bmatrix} M(q) & J^{t} \\ J & \mathbf{O} \end{bmatrix} \begin{bmatrix} \ddot{q} \\ \lambda \end{bmatrix} = \begin{bmatrix} Q(\dot{q}, q) \\ v(\dot{q}, q) \end{bmatrix}, \tag{2.9}$$

$$\phi(q) = 0, \quad \dot{\phi} \triangleq J\dot{q} = 0.$$

From the second derivative of ϕ we have $J\ddot{q} - v(\dot{q}, q) = 0$. Under reasonable assumptions on M and J, the linear system in $\ddot{q}$ and λ defined by (2.9) has a unique solution $\ddot{q} = f_1(\dot{q}, q)$; $\lambda = f_2(\dot{q}, q)$. If we let $\dot{q} = v$, $x = (q, v)^t$ then we

have $\dot{q} = v$; $\dot{v} = f_1(\dot{q}, q)$, together with $\phi(q) = 0$; $Jv = 0$. The above equations can be written in the form:

$$\dot{x} = f(x), \tag{2.10}$$

$$g(x) = 0, \tag{2.11}$$

respectively. Eq. (2.10) defines a vector field on the constraint manifold defined by (2.11) and it has some special structure which should be exploited while implementing. In Ref. [14] equation of motion for CMS is derived using D'Alembert principle.

Each f-evaluation involves the solution of the linear system (2.9), whose coefficient matrix has a nice (2×2) block structure. In most practical examples in CMS, the matrices $M(q)$ and $J(q)$ are sparse. Special techniques like block factorization, parallel processing techniques and sparse matrix algorithms should be employed. For large systems it is advantageous and preferable to use iterative methods i.e., solving the linear system as an optimization problem [15,16]. Also note that $\dot{\phi}$ is linear in v, and ϕ does not involve v. These facts can be nicely utilized during the numerical solution of the vector field.

(b) Simulation of control systems modeled by DAEs: When deriving models of physical systems from first principles, the result is often a system of DAEs (written in first-order form) of the type,

$$F(\dot{x}, x) = 0, \tag{2.12}$$

where $F: \mathbb{R}^n \to \mathbb{R}^n$. Eq. (2.12) is the most reasonable and user convenient representation for model libraries. If (2.12) is such that $\dot{x}$ can be solved as a function of x then (2.12) is simply an ODE. If not, the complexity of numerically solving such DAEs is measured by a positive integer quantity called the index. Good numerical methods are available for systems of index one [17]; in fact an excellent and popularly used code called DASSL [18] is available for solving such systems. For systems whose index is greater than one, good general purpose numerical methods are not well discussed in Ref. [6].

A DAE is equivalent to a vector field. However, a difficult process called index reduction is needed to convert a DAE of the form (2.12) (if it is of higher order) to the form (1.1), (1.2). There is an approximation of this process called structural index reduction, due to Panteleides [19] which is very simple and efficient. Also, on a number of DAEs arising from practical control system models, structural index reduction does give proper index reduction. Thus, for such DAEs, structural index reduction combined with the approaches discussed in this paper yields neat as well as valid solutions.

(c) Handling physical invariants: For many physical systems whose dynamic equations are of the form (2.2), the physics of the system dictates that certain physical quantities, such as the net charge or the total energy, be conserved, i.e., an equation of the form $c(x) = 0$, is satisfied where $c: \mathbb{R}^n \to \mathbb{R}$. In many systems, the satisfaction of $c(x) = 0$ by the numerical solution is important

because the solution can lead to erroneous behavior otherwise. Thus all the crucial conservation laws of the form $c(x) = 0$ must be collected to form (2.1) so that, when (2.1), (2.2) is numerically solved the conservation law is always satisfied.

(d) Constrained optimization: Consider the problem,

$$\min p(x) \quad \text{s.t. } g(x) = 0, \tag{2.13}$$

where $p: \mathbb{R}^n \to \mathbb{R}$ and $g: \mathbb{R}^n \to \mathbb{R}^m$ are continuously differentiable functions. Let $\mathcal{M} = \{x: g(x) = 0\}$. For $x \in \mathcal{M}$, define $f(x)$ to be the projection of $-\nabla p(x)$ on to $T_x\mathcal{M}$. The gradient projection method, is a good way of solving (2.13). One iteration of this method consists of using a recurrence of the form

$$x_{k+1} = x_k + h_k f(x_k) \tag{2.14}$$

and then correcting x_{k+1} so as to make $x_{k+1} \in \mathcal{M}$. A continuous realization of (2.14) is (2.2). Thus the vector field, (2.1), (2.2) results. Under reasonable assumptions it can be shown that, the solution of (2.2) starting from a random $x_0 \in \mathcal{M}$ converges to a local minimum of (2.13) with probability one. Thus a numerical solution of the vector field can be effectively used to find a local minimum of (2.13).

3. Existing approaches

The three main approaches for numerically solving vector fields are the following: Parameterization, Constraint Stabilization, and Perturbation. In this section we give a self-contained discussion of these approaches. In the process, we make two useful contributions: (1) for the Parameterization approach, we suggest a simple way of doing proper integration error control of the system variables; and (2) for the PA, we give a much more expanded analysis than that given by Shampine [20]. We begin this section by discussing a common tool that is used by all the three approaches: solution of a system of nonlinear equations.

3.1. Solution of nonlinear equations

Most of the approaches for solving vector fields require a procedure that finds a root of a square nonlinear system of equations the form, $h(z) = 0$, where $h: \mathbb{R}^m \to \mathbb{R}^m$. For example such a problem is encountered when x is restricted to an m-dimensional affine space and (2.1) is to be solved. We will see later that, in the PA, the manifold correction after each integration step involves the solution of nonlinear equations of the above form and in the case of Parameterization and Exact constraint stabilization approaches (ECS), every derivative function evaluation involves the solution of nonlinear equations. Usually z^0,

a very good guess of a root z^*, is available. Therefore, a locally convergent method such as the Newton–Raphson method will work well when started from z^0. Since the evaluation of $h_z(z)$ and its LU decomposition are expensive one uses the modified Newton–Raphson method (MNR) that uses the following iteration:

$$z^{l+1} = z^l - [h_z(z^0)]^{-1} h(z^l), \quad l \geqslant 0. \tag{3.1}$$

It can be shown [21] that this method is locally convergent, but with a linear rate of convergence. This is bad when compared to the nice quadratic rate of convergence of the Newton–Raphson method. However, this slight loss is made up by the savings made in avoiding the evaluation of $h_z(z^l)$ and its LU or $l \geqslant 1$. Let us rewrite (3.1) as

$$A\ \delta z = -h(z^l), \tag{3.2}$$

where $A = [h_z(z^0)]$ and $\delta z = z^{l+1} - z^l$. Eq. (3.2) is a linear system and it is usually solved by computing the LU decomposition of A.

The termination of the iteration process (3.1) is a very important factor to the reliability and robustness of the software. We now propose a simple scheme for the termination of the iteration process (3.1). Let us first define some important parameters to be used in the scheme. Let: typ z be an m-dimensional vector whose ith component is a positive scalar specifying the typical magnitude of z_i, $z \in \mathbb{R}^m$, typ h be an m-dimensional vector whose ith component is a positive scalar specifying the typical magnitude of the ith component of the function $h(z)$ at points that are not near the root of $h(z)$. The vector typ h is used to determine the scale factor vector S_h and the diagonal scaling matrix $D = \operatorname{diag}[(S_h)_1, \ldots, (S_h)_m]$, where $(S_h)_i = 1/(\operatorname{typ} h_i)$, $i = 1, \ldots, m$. Now $Dh(z^l)$ is a dimensionless vector whose components have the same order of magnitude. We choose a positive scalar tolerance, $h$tol and terminate the MNR iterations when

$$\|Dh(z^l)\|_\infty \leqslant h\,\text{tol}. \tag{3.3}$$

This is the basic idea behind the first scheme. The automatic selection of typ h and h tol is explained next.

Let $h_i(z^l)$ denote the ith component of $h(z^l)$ and let a^i denote the transpose of the ith row of $h_z(z^0)$. Then a good estimate of typ h_i, the typical size of h_i, is given by

$$\operatorname{typ} h_i = \max_{1 \leqslant j \leqslant m} \left(|a_j^i| \max\{|z_j^0|, \operatorname{typ} z_j\} \right). \tag{3.4}$$

If typ z is not available then we can simply replace (3.4) by

$$\operatorname{typ} h_i = \max_{1 \leqslant j \leqslant m} \left(|a_j^i|\ |z_j^0| \right). \tag{3.5}$$

The only difficulty with (3.5) is that it is meaningless when $z^0 \approx 0$. We now calculate $\|Dh\|_\infty$ as

$$\|Dh\|_\infty = \max_{1 \leqslant i \leqslant m} \left| \frac{h_i}{\max_{1 \leqslant j \leqslant m}(|a_j^i| \max\{|z_j^0|, \operatorname{typ} z_j\})} \right|.$$

For best accuracy, a good choice for h tol is

$$h\,\text{tol} = u \,.\, v, \tag{3.6}$$

where u is the unit round off error of the computing machine used and v is factor included to take care of errors in computing $h_i(z^l)$; a value of $v = 100$ m is a reasonable choice [22]. An excellent discussion on floating representation, floating arithmetic and rounding errors is given in Ref. [23]. For computing the machine constants like u, unit round off, base etc. for different computer systems, a code called MACHAR [24] is available.

Since z^0 is a very good guess of a solution to $h(z) = 0$, it is a good idea to check first if (3.3) holds for $l = 0$. If it holds, then the computation of $[h_z(z^0)]^{-1}$ is unnecessary. Also, if convergence according to (3.3) does not take place till $l = 4$ (MAXITER), the method should be terminated as having failed and other measures, which yield an improved z^0 have to be taken.

One could also employ an alternative termination scheme, which is much more detailed than the first. A good test of convergence of the MNR method, for use in the solutions of stiff ordinary differential equations was suggested by Shampine [25]. An MNR scheme implementing this test incorporates the integration error tolerances into the iteration termination process. Shampine's idea has also been implemented in the well-known DAE software, DASSL [18]. This alternative scheme is nicely explained in Ref. [17]. It is more detailed and expensive than our scheme. We have experimented in detail with both schemes and have found both to be equally good.

3.2. The parameterization approach

The parameterization approach, initiated by Wehage and Haug [26], has been popularly used for solving CMS. To describe the approach let us begin with an ideal situation. Suppose a global parameterization of $\mathcal{M}$ is available. Then a numerical solution of (2.2) and a simultaneous enforcement of (2.1) is easy. Let $\psi: \mathbb{R}^{n-m} \to \mathcal{M}$ be a known diffeomorphism (global parameterization) and $\psi(y_0) = x_0$. Take any $y \in \mathbb{R}^{n-m}$ and let $x = \psi(y)$. Then $\psi_y(y)$, the $n \times (n-m)$ Jacobian of ψ evaluated at y, defines an isomorphism from $\mathbb{R}^{n-m}$ to $T_x\mathcal{M}$. Let $\Gamma(\cdot; y): T_x\mathcal{M} \to \mathbb{R}^{n-m}$ denote the inverse of this isomorphism. Procedure-wise, given v, $w = \Gamma(v; y)$ can be obtained by solving any one set of $(n-m)$ linearly independent equations chosen from the system $\psi_y(y)w = v$. Then $x(\cdot)$ is obtained by solving the ODE, $\dot{y} = \Gamma(f(\psi(y)); y)$, $t \in [t_0, t_f]$, $y(t_0) = y_0$, and setting $x(t) = \psi(y(t))$, $t \in [t_0, t_f]$ and $x(t) \in \mathcal{M}$ is assured for all t.

In most applications where (2.1), (2.2) is encountered, either a global parameterization does not exist, or, it exists but it is unknown. In that case, a reasonable approach is to resort to a numerically computed local parameterization of $\mathcal{M}$ around x_0. Since it is important for implementation purposes, let us go through the details.

Let $T = [U_0|V_0]$ be an $(n \times n)$ orthogonal matrix such that $U_0 \in \mathbb{R}^{n\times(n-m)}$, $V_0 \in \mathbb{R}^{n\times m}$ and $P(x_0) = [g_x^t(x_0)|U_0]^t$ is nonsingular. Later we will describe two specific choices for U_0 and V_0 which lead to two different parameterization approaches. If g_x is a continuous function of x then there exists X, a neighborhood of x_0 such that

$$P(x) = \begin{pmatrix} g_x(x) \\ U_0^t \end{pmatrix} \text{ is nonsingular } \quad \forall x \in X.$$

Since $\operatorname{rank}(P(x)V_0) = m$ and $U_0^t V_0 = 0$, we have $g_x(x)V_0$ is an $m \times m$ *nonsingular matrix* $\forall x \in X$. Now consider the transformation of variables, $(y,z)^t == T^t(x - x_0)$. The inverse transformation is $x = x_0 + U_0 y + V_0 z$.

In terms of the new variables (2.1) can be expressed as $0 = \tilde{g}(y,z) \triangleq g(x_0 + U_0 y + V_0 z)$. Let $\tilde{\mathcal{M}} = \{(y,z): \tilde{g}(y,z) = 0\}$. Now

$$\tilde{g}_z(y,z) = g_x(x_0 + U_0 y + V_0 z)V_0. \tag{3.7}$$

It is clear that $\tilde{g}_z(0,0) = g_x(x_0)V_0$ is an $m \times m$ nonsingular matrix. By the implicit function theorem, y provides a local parameterization of $\tilde{\mathcal{M}}$. In other words, there exists a neighborhood Y around the origin in $\mathbb{R}^{n-m}$, a neighborhood $\tilde{X}$ around the origin in $\mathbb{R}^n$ and a diffeomorphism $\tilde{\psi}: Y \to \tilde{X}$ such that: (i) $\tilde{\psi}(0) = 0$; and (ii) $(y,z) \in \tilde{\mathcal{M}} \cap \tilde{X} \iff (y,z) = \tilde{\psi}(y)$.

Since (y,z) and x are related by the invertible affine transformation, y also provides a local parameterization of $\mathcal{M}$. The diffeomorphism, $\psi: Y \to X$ defined by $X = x_0 + T\tilde{X}$ and $\psi(y) = x_0 + T\tilde{\psi}(y)$, is a local parameterization of $\mathcal{M}$ at x_0. Procedure-wise, the computation of $x = \psi(y)$ given $y \in Y$ is done in two steps: (a) solve $\tilde{g}(y,z) = 0$ for z such that $(y,z) \in \tilde{X}$; and (b) set $x = x_0 + U_0 y + V_0 z$. These steps are equivalent to solving the following nonlinear system for $x \in X$:

$$g(x) = 0, \quad U_0^t(x - x_0) - y = 0. \tag{3.8}$$

It is now easy to set up an ODE for y in Y:

$$\dot{y} = U_0^t \dot{x} = U_0^t f(x) = U_0^t f(\psi(y)) \triangleq \tilde{f}(y). \tag{3.9}$$

Once (3.9) is solved with $y(t_0) = 0$, we can find $x(\cdot)$ from

$$x(t) = \psi(y(t)). \tag{3.10}$$

After setting the ODE in y we now turn our attention on integrating it numerically. Integration of the ODE (3.9), involves the evaluation of the right-hand side function $\tilde{f}(y)$. It is worthwhile to study the computation of $\tilde{f}(y)$. The following procedure computes $\tilde{f}(y)$ for given y.

Procedure TP. Computing $\tilde{f}(y)$ given y.
Step 1: Solve $\tilde{g}(y,z) = 0$ for z;
Step 2: Set $x = \psi(y) = x_0 + U_0 y + V_0 z$ (diffeomorphism);
Step 3: Compute $f(x)$ and set $\tilde{f}(y) = U_0^{\mathrm{t}} f(x)$.

Two important observations have to be made here. First, (3.10) does not have to be carried out as a separate operation; rather, it comes as a by-product when $\tilde{f}(y)$ is evaluated. Second, each evaluation of $\tilde{f}(y)$ is somewhat expensive because, apart from one evaluation of $f(x)$ it requires the solution of the nonlinear system, $\tilde{g}(y,z) = 0$ for the m-dimensional vector, z. Due to this reason, integration methods which utilize fewer derivative function evaluations should be used to solve (3.9). In particular, when dealing with nonstiff systems, Adams methods which require only about 2 $\tilde{f}$ evaluations per integration step should be preferred over Runge–Kutta methods.

The merits of the Parameterization approach are that, it fundamentally includes and enforces (2.1), it involves the integration of only the $(n-m)$ dimensional ODE system, (3.9), and, it involves the evaluation of $f(x)$ only at points, $x \in \mathcal{M}$. Next we mention its defects in the following three remarks.

Remark 3.1. As already mentioned, each evaluation of $\tilde{f}(y)$ is expensive because it requires the solution of the nonlinear system, $\tilde{g}(y,z) = 0$. Since the evaluation of $\tilde{g}_z(y,z)$ and its factorization are usually expensive, this system should be solved using the MNR method. For this method to work well, a good starting iterate for z must be provided. Later, when we discuss specific parameterization approaches, we will see how this is done.

Remark 3.2. To avoid the resetting of the ODE in (3.9), the parameterization done at x_0 is continued for as many integration steps as possible. But a change in parameterization becomes a necessity when x crosses out of the range of parameterization, X as integration progresses. This usually occurs when the nonsingularity condition is violated. If a single step integration method, such as a Runge–Kutta method, is used to solve (3.9), the change in parameterization is not a serious issue. It is, however, an important factor when a multistep method such as an Adams method is used. A multistep method derives its efficiency by changing the orders of integration formulas. Large integration steps are usually taken at high orders. Suppose a change in parameterization becomes necessary while the method is operating with a high order integration formula, since the ODE in (3.9) has to be altered, integration has to be restarted from a first order formula, leading to a severe loss of efficiency.

Remark 3.3. Interpolation on x, required for purposes such as graphical output and root finding, is somewhat expensive when the Parameterization approach is used. This is because integration methods only yield an interpolant for y and,

obtaining x for a given y requires the execution of Steps 1 and 2 of Procedure TP.

How should U_0 and V_0 be chosen? It will be ideal to choose them in such a way that (3.9) is, in some sense, nicely formed so that its numerical integration can be done more efficiently than that of (2.2). But, with all popular integration methods, e.g., Runge–Kutta and linear multistep methods, integration is unaffected by an affine transformation of the variables. In other words, suppose $x = T\bar{x} + s$ is a transformation where T is nonsingular. Then, given $\dot{x} = f(x)$, carrying out numerical integration on x, and, carrying out the same on $\dot{\bar{x}} = T^{-1}f(T\bar{x} + s)$ and then setting $x = T\bar{x} + s$, will yield identical solutions; of course, we are assuming that T is appropriately included in the error measurement of $\bar{x}$. From the definition of $x(t)$, integration efficiency is unaffected whatever be the choice of U_0 and V_0. Therefore, the choice should be based on other factors such as improving the efficiency of transition when a change in parameterization is needed, and, making X, the region of parameterization as large as possible.

Two specific choices, termed 'Coordinate Partitioning' (CP) and 'Tangential Parameterization' (TP), have been popular. In the CP approach, due to Wehage and Haug [26], $T = [U_0|V_0]$ is chosen to be a permutation of the identity matrix. In other words, (y, z) is nothing but a reordering of $(x - x_0)$ such that g_z is nonsingular at x_0. This reordering can be decided by doing an LU factorization of $g_x(x_0)$ using column pivoting. In CP, an extrapolator is maintained on z. This extrapolator is a polynomial of the same degree as that of the interpolant for y used in integrating (3.9). The extrapolator on z is used for two purposes: (1) it provides a good starting iterate for z when $\tilde{g}(y, z) = 0$ is solved by the MNR method; and (2) when a multistep integration method is used and the change in parameterization mentioned in Remark 3.2 becomes necessary, the extrapolator on z can be used together with the interpolant on y to start the integration of the new ODE system at the same order as that used in the last integration step of the old ODE system. The use of the extrapolator for the second purpose certainly improves efficiency. However, from a stringent accuracy point of view, the use of the extrapolator for restarting integration should be objected to, because the extrapolator was not based on error control.

The idea of TP was introduced by Mani et al. [27]. Here V_0 is chosen so that its column space is the same as that of $g_x^t(x_0)$. In other words, the columns of U_0 form an orthogonal basis for $T_{x_0}\mathcal{M}$. U_0 and V_0 can be computed by doing a full QR decomposition of $g_x^t(x_0)$. There are two motivating reasons for the above choice of U_0 and V_0. First, given only the first-order data of g at x_0, one expects that X, the region of validity of the local parameterization corresponding to the above choice of U_0 and V_0 is "bigger" than that of any other choice. This means that this approach will require fewer changes in parameterization. Second, suppose integration of (3.9) is being advanced from y_k to y_{k+1} in a time step.

Whenever $\tilde{g}(y,z) = 0$ has to be solved for z during this time step, $z = z_k$ provides an excellent starting iterate for the MNR method.

For TP, Mani et al. [27] use an interesting idea to decide when a change in parameterization is needed. Roughly, their idea is as follows. At x_0, it should be clear that $\|\dot{y}\|_2 = \|\dot{x}\|_2$ and $\|\dot{z}\|_2 = 0$. The ratio, $\|\dot{y}\|_2/\|\dot{x}\|_2$ gives a good indication of a need for change in parameterization. Choosing an appropriate $\mu \in (0,1)$, say $\mu = 0.5$, a change in parameterization is called for when $\|\dot{y}\|/\|\dot{x}\| < \mu$ occurs.

Let us look at some factors of comparison between CP and TP. Usually changes in parameterization are less frequent when TP is used. This is to be expected. However, when a multistep method is employed for integration the inefficiency caused by a change in parameterization is very severe for TP because it has to restart integration from a first-order formula; recall Remark 3.2. One heuristic for restarting is to use the $(x,\dot{x})$ values of the past integration grid points to generate the $(y,\dot{y})$ values corresponding to the new parameterization at these points, and build an interpolant of appropriate order for $(y,\dot{y})$ to restart integration efficiently. Even this modification involves a lot of overhead. Thus, if integration involves a small motion that does not require a change in parameterization, TP is more efficient. On the other hand, if there is a large motion that requires several changes in parameterization even for TP then CP will perform better. The popular code, DADS [28] which solves CMS, uses a modification of CP [29].

Integration error control: An important issue concerning error control has not been carefully addressed in the literature. To explain and address the issue we need to discuss how local error control is done in one integration step. At $t = t_k$ let: y_k denote the approximate y given by numerical integration; and $x_k = \psi(y_k)$. When stepping from t_k to some $t_{k+1} = t_k + h_k$ using local error control, y_k and x_k are assumed to be exact. Let $y(\cdot)$ and $x(\cdot)$ denote the true local solution, i.e., solutions of (3.9) and (3.10) with $y(t_k) = y_k$. Integration formulas are used to obtain y_{k+1} and x_{k+1}, which are approximations of $y(t_{k+1})$ and $x(t_{k+1})$, respectively. Define the error vectors as: $e_x = x_{k+1} - x(t_{k+1})$ and $e_y = y_{k+1} - y(t_{k+1})$. A scalar measure of error is usually specified by the user of the integration code as an appropriate norm on e_x. Popular codes using single step integration methods employ a scaled l_∞-norm while most codes based on multistep methods use a scaled l_2-norm. Let us take the latter as an example. The aim, then, is to choose the step size h_k large enough so that

$$\|e_x\|_W = \|x_{k+1} - x(t_{k+1})\|_W = \|\psi(y_{k+1}) - \psi(y(t_{k+1}))\|_W \leqslant \tau, \tag{3.11}$$

where τ is a specified tolerance and W is usually a diagonal matrix that specifies variable weights for the components of x and $\|x\|^2 = x^t W x$. The difficulty is that (3.11) is not a direct norm specification on e_y because ψ is a nonlinear function of y. Fortunately, a reasonable approximation helps us to overcome the problem.

The first-order Taylor series approximation of ψ around y_{k+1} gives

$$\psi(y(t_{k+1})) - \psi(y_{k+1}) \approx \psi_y(y_{k+1})(y(t_{k+1}) - y_{k+1}). \tag{3.12}$$

This is certainly a good approximation because $y(t_{k+1})$ and y_{k+1} are near to each other. Using this in (3.11) we get a reasonable approximation for $\|e_x\|_W$ as

$$\|e_x\|_W \approx \|e_y\|_{\tilde{W}}, \tag{3.13}$$

where $\tilde{W} = \psi_y^t(y_{k+1}) W \psi_y(y_{k+1})$.

Remark 3.4. One may object to the use of $\psi_y(y_{k+1})$ because, when h_k is being decided y_{k+1} is unknown. In most codes, however, h_k is chosen based on information from previous integration steps and then, a check is made whether the error is within the tolerance. If the error is within the tolerance, this h_k is accepted; else, it is reduced. In such a mode of operation the use of $\psi_y(y_{k+1})$ is acceptable. In some codes, such as those based on Taylor series methods, the choice of h_k is based on estimating $\|e_x\|_W$. In such cases, the best remedy is to assume that ψ_y will remain fairly unchanged in the integration step and use $\psi_y(y_k)$ instead of $\psi_y(y_{k+1})$ in (3.13).

How do we compute $\psi_y(y_{k+1})$? Since $x = \psi(y)$ is the solution of (3.8), it is easy to differentiate that system with respect to y and obtain

$$\psi_y(y_{k+1}) = \begin{bmatrix} g_x(x_{k+1}) \\ U_0^t \end{bmatrix}^{-1} \begin{pmatrix} 0 \\ I_{n-m} \end{pmatrix}. \tag{3.14}$$

This is not an efficient way of computing $\psi_y(y_{k+1})$. A more useful formula, which can be verified using (3.14) is

$$\psi_y(y_{k+1}) = \left[I_n - V_0 (g_x(x_{k+1}) V_0)^{-1} g_x(x_{k+1}) \right] U_0. \tag{3.15}$$

Pre-multiplying (3.15) by $g_x(x_{k+1})$ will give $g_x(x_{k+1})\psi_y(y_{k+1}) = 0$ and pre-multiplying it by U_0^t will give $U_0^t \psi_y(y_{k+1}) = I_{n-m}$. A good approximation of the inverse (in the sense of LU decomposition) required in the above formula is available from prior computations. To see this, note that $\tilde{g}(y_{k+1}, z) = 0$ is solved for z_{k+1} by the MNR method, so as to obtain x_{k+1}. This requires the inverse of $\tilde{g}(y_{k+1}, \bar{z})$ where $\bar{z}$ is the starting iterate for the MNR iterations. By (3.7) and the fact that $\bar{z}$ and z_{k+1} are near to each other, the inverse of $\tilde{g}_z(y_{k+1}, \bar{z})$ is a good approximation of the inverse of $g_x(x_{k+1}) V_0$.

3.3. The constraint stabilization approach

The basic idea of this approach is to replace the solution of (2.1), (2.2) by the output-stabilization of an associated nonlinear dynamic system with control, $u \in \mathbb{R}^m$ and output, $y \in \mathbb{R}^m$:

$$\dot{x} = f(x) + P(x)u, \tag{3.16}$$

$$y = g(x), \tag{3.17}$$

where $P: \mathbb{R}^n \to \mathbb{R}^{n\times m}$ is a matrix function that satisfies the following assumption.

Assumption 3.1. $\exists$ an open set $\mathcal{O}$ in $\mathbb{R}^n$ containing $\mathcal{M}$ such that $g_x(x)P(x)$ is nonsingular $\forall x \in \mathcal{O}$.

The aim is to use u to regulate the output y in a neighborhood of $\mathcal{M}$. On $\mathcal{M}$, exact regulation, i.e., $y(\cdot) = \mathbf{0}$ ($\mathbf{0}$ denotes the identically zero function) is possible. The following simple theorem, which is only a slight generalization of a result, due to Gear [30], establishes that if u is chosen to accomplish exact regulation on $\mathcal{M}$ then (3.16) and (2.2) are identical.

Theorem 3.1. *Let $x_0 \in \mathcal{M}$. Given $u(\cdot)$, let $y(\cdot)$ denote the solution for y obtained by solving (3.16),(3.17) with $x(t_0) = x_0$. Then $y(\cdot) = \mathbf{0}$ iff $u(\cdot) = \mathbf{0}$. In other words, if $x_0 \in \mathcal{M}$ and the $u(\cdot)$ is chosen in (3.16) to do exact regulation, i.e., force (2.1), then the solutions of (3.16) and (2.2) starting from $x(t_0) = x_0$ are identical.*

Proof. The if part is trivial because (2.2) is a vector field on $\mathcal{M}$. The only if part is also easy. Since $y(\cdot) = \mathbf{0}$, $\mathbf{0} = \dot{y} = g_x(x)\dot{x} = g_x(x)f(x) + g_x(x)P(x)u$. Since

$$g_x(x)f(x) = 0 \quad \forall x \in \mathcal{M}, \tag{3.18}$$

and $g_x(x)P(x)$ is non-singular, $u = \mathbf{0}$. □

There are two very different ways of using (3.16) and (3.17) for numerically solving (2.1) and (2.2). These are due to Baumgarte and Gear.

3.3.1. Inexact constraint stabilization approach

In Baumgarte's approach [31],

$$P(x) = g_x^t(x) \tag{3.19}$$

is chosen and a feedback law, $u = K(x)$ is designed so that y is exponentially regulated. For example, we can choose a positive-definite diagonal matrix, D and require that y satisfies

$$\dot{y} + Dy = 0. \tag{3.20}$$

Using (3.16)–(3.19) it is easy to verify that the unique choice of u that leads to (3.20) is

$$u = K(x) = -[g_x(x)g_x^t(x)]^{-1}Dg(x). \tag{3.21}$$

Putting this in (3.16) gives

$$\dot{x} = f(x) + g_x^t K(x) \triangleq \hat{f}(x). \tag{3.22}$$

Since $K(x) = 0\ \forall x \in \mathcal{M}$, (3.22) is an extension of (2.2). For $x \notin / \mathcal{M}$, $\hat{f}(x)$ roughly corresponds to a modification of $f(x)$ so that the derivative vector points towards $\mathcal{M}$.

Baumgarte's approach consists of setting up (3.22) and then applying a numerical ODE solver on it. This approach is the simplest of all approaches available for solving (2.1), (2.2). However there are three issues that limit its usefulness. First, the structure of the solution trajectories of (3.22) is not clear. This has to be understood if proper error control is to be done.

The second issue concerns the choice of stabilization parameter D. If D is chosen to have large numbers so as to stabilize y fast, then (3.22) may become a stiff system even if (2.2) is nonstiff. Thus, a lot of care is needed in choosing D. Baumgarte [34] suggests that the choice of D should be done adaptively. But a clear, general way of choosing D is not so far available at all.

The third issue is regarding efficiency. By (3.21), each evaluation of $\hat{f}(x)$ requires the evaluation of $g_x(x)$ and the solution of the linear system, $[g_x(x)g_x^t(x)]u = -Dg(x)$. The best way to solve this system is via a skinny QR decomposition of $g_x^t(x)$. These matrix operations are roughly the same as those done by the PA in the correction-to-manifold phase (Step 2 of Procedure PA). Thus, simplicity of Baumgarte's approach should not be misunderstood to mean efficiency. In fact, there is a good possibility that the PA is more efficient than Baumgarte's approach that uses (3.22). To improve efficiency, we could replace (3.21) by $u = -Dg(x)$. This does lead to the stable system, $\dot{y} + g_x(x)g_x^t(x)Dy = 0$; however, in this case it is difficult to choose D so that y has a prescribed exponential rate of decay. For CMS, a special setup allows Baumgarte's approach to be very efficient; in fact, Baumgarte's initial motivation comes from these systems. An extension of this approach to the slightly more general case of index-3 in Hessenberg form is discussed in [32].

3.3.2. Exact constraint stabilization approach

Gear's approach [30,33] can be viewed as a complicated discretization of (3.16), (3.17) using an integration formula where u is chosen to do a deadbeat control of y wherever it is required. The aim is to involve (2.1) directly in the numerical solution of (2.2). As mentioned earlier the direct numerical solution that simultaneously takes care of both (2.1) and (2.2) is not easy because they form an over-determined differential-algebraic system, though they are consistent. The introduction of u and the replacement of (2.2) by (3.16) provide the additional variables necessary to make the system fully determined. By Theorem 3.1 the system (3.16), (2.1) is well defined and, on $\mathcal{M}$ it is identical with the original system (2.2). However, numerical solutions of the two systems are different. A numerical solution of (3.16) and (2.1) ensures (2.1) at the cost of violating the invariant,

$$u = 0$$

i.e., altering (2.2). On the other hand, a numerical solution of (2.2) does not guarantee (2.1). Gear's approach consists of solving the index-2 differential-algebraic equation system defined by (3.16) and (2.1) with $P(x) = g_x^t(x)$ and this system is called *stabilized index two* problem. The following procedure describes the basic steps involved in an integration step from time t_n to t_{n+1}.

We will now discuss the implementation details of ECS. Before going into details, let us first briefly discuss about backward difference formula (BDF) method mentioned in the algorithm. The formula used is the fixed leading form of the kth order BDF method. Stability of BDF methods applied to index-2 DAEs is discussed in [34]. Block BDF methods for DAEs are discussed in [35].

The values of the predictor $x_{n+1}^p, \dot{x}_{n+1}^p$ and the corrector x_{n+1} at t_{n+1} are defined in terms of polynomials which interpolate the solution at previous time points. Following the ideas of Krogh [36] and Shampine and Gordon [37] these polynomials are represented in terms of modified divided difference. The corrector formula can be written as [38]

$$\dot{x}_{n+1} = \dot{x}_{n+1}^p - \left(\frac{\alpha_s}{h_{n+1}}\right)(x_{n+1} - x_{n+1}^p),$$

where the fixed leading coefficient α_s is defined as $\alpha_s = -\sum_{j=1}^{k}\left(\frac{1}{j}\right)$.

Procedure ECS. Stabilized index two formulation.

Step 1: Predict x at time t_{n+1} from the previous values and obtain x_{n+1}^p.

Step 2: Evaluate $f(x_{n+1}^p)$ and $g_x(x_{n+1}^p)$.

Step 3: Using an appropriate BDF for $\dot{x}$, solve for u_{n+1} by MNR iteration from (2.1).

Step 4: Repeat steps 2 to 3 once, using the latest computed values of x_{n+1} for x_{n+1}^p. This amounts to a single corrector iteration.

The specialization of this basic idea to CMS equations will be discussed in the forthcoming paper [39].

3.4. The perturbation approach

In the context of maintaining conservation laws of ODEs, Shampine [20] suggested that the solution be perturbed after each integration step so as to satisfy the conservation laws. This is essentially a new way of solving vector fields, and we have called it as the Perturbation approach. In this section we give a much more detailed analysis of the PA than that given by Shampine. The new approach overcomes some of the difficulties associated with the other two approaches. Its salient features are: (1) it decouples the process of integration from the process of correction to the constraint manifold; (2) it solves the vector field in terms of the original coordinates and hence the integration

tolerances for the original variables can be specified directly; and (3) it does not involve any coordinate transformation which has to be carried out in some of the other approaches.

In the PA, a correction is applied to a numerical solution of (2.2) after each integration step so as to satisfy (2.1). To describe the approach, it is sufficient to say what is done in one integration step. Suppose k steps of the numerical solution of (2.1), (2.2) have been done and $t = t_k$ has been reached. Let $x_k \in \mathcal{M}$ be the solution approximant at $t = t_k$. Denote the local solution by $x(\cdot)$, i.e., $x(\cdot)$ is the solution of (2.2) with $x(t_k) = x_k$. Let τ denote the integration tolerance. In the $(k+1)$st step, the aim is to determine a step size h_k and an $x_{k+1} \in \mathcal{M}$ that satisfy

$$\|x(t_{k+1}) - x_{k+1}\| \leqslant \tau, \tag{3.23}$$

where $t_{k+1} = t_k + h_k$. The determination of h_k and x_{k+1} is described by the following procedure.

Procedure PA. Determination of h_k and x_{k+1} by the PA.

1. Numerically integrate (2.2) from $x(t_k) = x_k$ using local error control (without taking into account (2.1)) to obtain a step size h_k and an approximant, $\bar{x}_{k+1}$ that satisfy

$$\|x(t_{k+1}) - \bar{x}_{k+1}\| \leqslant \tau/2. \tag{3.24}$$

2. Solve the optimization problem

$$\min \ \|x - \bar{x}_{k+1}\| \quad \text{s.t. } g(x) = 0 \tag{3.25}$$

and set x_{k+1} = the minimizer of (3.25).

The following theorem establishes the correctness of the procedure.

Theorem 3.2. *The x_{k+1} determined by Procedure PA satisfies $x_{k+1} \in \mathcal{M}$ and (3.23).*

Proof. Since x_{k+1} is feasible for (3.25), it satisfies $x_{k+1} \in \mathcal{M}$. Consider $x(t_{k+1})$, the true local solution of (2.2) at t_{k+1}. Since $x(t) \in \mathcal{M}$ $\forall t$, $x(t_{k+1})$ is feasible for (3.25). By optimality, $\|x_{k+1} - \bar{x}_{k+1}\| \leqslant \|x(t_{k+1}) - \bar{x}_{k+1}\|$. Using this together with (3.24) and the triangle inequality we get

$$\begin{aligned} \|x_{k+1} - x(t_{k+1})\| &\leqslant \|x_{k+1} - \bar{x}_{k+1}\| + \|x(t_{k+1}) - \bar{x}_{k+1}\| \\ &\leqslant 2\|x(t_{k+1}) - \bar{x}_{k+1}\| \leqslant \tau. \end{aligned} \tag{3.26}$$

□

Step 1 of Procedure PA can be carried out using any well-known numerical method for solving ODEs. Step 2 requires a subprocedure that solves (3.25). Before going in to the details of this subprocedure, let us make some useful remarks on the procedure and the approach.

Remark 3.5. Step 2 of Procedure PA is stronger than what is really needed. It is sufficient if x_{k+1} satisfies

$$g(x_{k+1}) = 0, \quad \|x_{k+1} - \bar{x}_{k+1}\| \leqslant \frac{\tau}{2}. \tag{3.27}$$

By the first inequality of (3.26) and (3.24), it follows that x_{k+1} also satisfies (3.23). Though the feasibility problem (3.27) is usually as difficult to solve as the optimization problem (3.25), it is useful because its solution is simpler to verify.

Remark 3.6. With all the popular integration methods, something more than an $\bar{x}_{k+1}$ satisfying (3.24) is available: an interpolant, $\bar{x}$: $[t_k, t_{k+1}] \to \mathbb{R}^n$ that satisfies $\bar{x}(t_k) = x_k$, $\bar{x}(t_{k+1}) = \bar{x}_{k+1}$, and, $\|\bar{x}(t) - x(t)\| \leqslant \tau/2 \ \forall t \in [t_k, t_{k+1}]$. For each t, define $\tilde{x}(t)$ to be the solution of

$$\min \|x - \bar{x}(t)\| \quad \text{s.t. } g(x) = 0. \tag{3.28}$$

Then, using the same ideas of the proof of Theorem 3.2, it is easy to establish

$$\tilde{x}(t) \in \mathcal{M}, \qquad \|\tilde{x}(t) - x(t)\| \leqslant \tau \ \ \forall t \in [t_k, t_{k+1}]. \tag{3.29}$$

Thus $\tilde{x}$ is a natural interpolant associated with PA. The evaluation of this interpolant is somewhat expensive since it requires the solution of (3.28) at each t.

Remark 3.7. In Step 1 integration is done using half the tolerance specified for the solution accuracy. This means that PA is inefficient when compared with the direct approach of simply integrating (2.2). This is a cost one has to pay for ensuring (2.1). It should be mentioned that halving the tolerance does not double the integration cost. This is because, all the popular integration methods use a formula of the form $\alpha h^{p+1} =$ tolerance, to determine the step size, where p is the order of integration formula used. If p_{ave} denotes the average order of formula used for integration, then we can roughly say that

$$\frac{\text{Number of integration steps for tolerance} = \tau/2}{\text{Number of integration steps for tolerance} = \tau} \approx r \triangleq (2)^{1/(p_{\text{ave}}+1)}.$$

Even if we take $p_{\text{ave}} = 4$, which is a reasonable value for good codes such as those in [4], $r = 1.149$. Thus, halving the tolerance increases the number of integration steps needed to do an integration task only by about 15%.

Now let us look at the possibility of replacing the integration tolerance, $\tau/2$ used in Step 1 by something bigger while ensuring that $x_{k+1} \in \mathcal{M}$ and (3.23) hold. There is something special when: the norm used in integration is a scaled l_2-norm, i.e., $\|x\| = \sqrt{x^t W x}$, where W is a symmetric positive definite matrix; and g is affine in the τ neighborhood of $\bar{x}_{k+1}$, i.e.,

$$g(x) = Ax + b \quad \forall x \ni \|x - \bar{x}_{k+1}\| \leqslant \tau\,. \tag{3.30}$$

Since x_{k+1} solves (3.25), $(y - x_{k+1})^t W(\bar{x}_{k+1} - x_{k+1}) = 0 \ \ \forall y \in \mathcal{M} \ni \|y - \bar{x}_{k+1}\| < \tau$. Thus

$$\|\bar{x}_{k+1} - x(t_{k+1})\|^2 = \|(\bar{x}_{k+1} - x_{k+1}) - (x(t_{k+1}) - x_{k+1})\|^2$$
$$= \|(\bar{x}_{k+1} - x_{k+1})\|^2 + \|(x(t_{k+1}) - x_{k+1})\|^2 \tag{3.31}$$

and hence,

$$\|x(t_{k+1}) - x_{k+1}\| \leqslant \|\bar{x}_{k+1} - x(t_{k+1})\|, \tag{3.32}$$

which means that x_{k+1} is more accurate than $\bar{x}_{k+1}$. In such a situation, then, it is permissible to replace the tolerance $\tau/2$ in (3.24) by τ.

The scenario is different when g is nonlinear. This nonlinearity of g can cause distortions which may not permit one to obtain nice bounds such as (3.32). However, we can say that, at stringent tolerances (3.30) is very nearly correct with $A = g_x(\bar{x}_{k+1})$, $b = g(\bar{x}_{k+1}) - g_x(\bar{x}_{k+1})\bar{x}_{k+1}$, and so the probability of (3.32) occurring is very high.

Remark 3.8. Shampine [20] cautions the use of a variable order multistep integration method, such as Adams method, in Step 1 of Procedure PA. His main objection is that, the perturbations of Step 2 may cause some 'roughness' in the interpolant of the multistep method, which in turn may affect the order changing mechanism. But, since the perturbations are structured (i.e., they are associated in a special way with the fundamental constraint (2.1), of the true solution), it is our belief that Shampine's caution is not serious. In a number of numerical tests carried out on several vector field examples arising from CMS, we have found that the order changing process associated with the solution obtained by the PA is not much different from that associated with the direct solution of (2.2). The Adams code of Shampine and Gordon [37] was used in these numerical tests. Specifically, we observed that the two order changing mechanisms are identical at low orders and, only slightly different at high orders. Furthermore, in spite of the small differences in the orders, the step sizes associated with the two solutions were nearly identical.

Let us now consider the solution of (3.25) required in Step 2 of Procedure PA. Usually, scaled l_2- and l_∞-norms are the popular norms used for measuring integration errors. Shampine suggests a method for solving (3.25) when the scaled l_2-norm is used. Further, he makes the remark that, "if the ODE solver is based on the maximum norm l_∞, one might well prefer to alter it to use the Euclidean norm l_2 so as to arrive at a linear algebra problem which has a classical solution". For the scaled l_2-norm we suggest a method which is slightly different but more efficient than Shampine's method. Furthermore, we also suggest a method for the scaled l_∞-norm and argue that, in some ways the computation of an l_∞-norm solution is cheaper than that of the l_2-norm solution.

Consider the scaled l_2-norm first. Here $\|x\| = \sqrt{x^t W x}$, where W is a symmetric positive definite matrix. Typically, W is a diagonal matrix, the diagonal

elements representing the variable weights applied to the different components of x. To simplify the notations, let us assume the objective function in (3.25) is replaced by half its square, and an appropriate coordinate transformation,

$$x = \bar{x}_{k+1} + TX \tag{3.33}$$

(T = a nonsingular matrix such that $T^t WT = I_n$) is done, so that (3.25) becomes

$$\min \ \|X\|_2^2/2 \quad \text{s.t. } G(X) = 0, \tag{3.34}$$

where

$$G(X) = g(\bar{x}_{k+1} + TX). \tag{3.35}$$

The first-order necessary conditions for (3.34) are

$$X - G_X^t(X)\mu = 0, \quad G(X) = 0. \tag{3.36}$$

where μ is the Lagrangian multiplier for equality constraint. We can use the MNR method to obtain X^*, a solution of (3.36).

Following Remark 3.5, finding an optimal solution of (3.34) is not crucial. This allows us to incorporate the following efficiency-improving modification: replace the term $G_X^t(X)$ in (3.36) by $G_X^t(0)$. Thus we solve

$$X - G_X^t(0)\mu = 0, \quad G(X) = 0, \tag{3.37}$$

instead of (3.36), using the MNR method to obtain a solution X^*.

Remark 3.9. Clearly, the X^* found above may not solve (3.34). If $\|X^*\| \leqslant \tau/2$, then we accept it and set $x_{k+1} = \bar{x}_{k+1} + TX^*$ for use in Step 2 of Procedure PA. This is because such an x_{k+1} satisfies (3.27), and, Remark 3.5 says this is sufficient for the success of Procedure PA. If $\|X^*\| > \tau/2$ then there are two alternatives. Either a more detailed method is used to solve (3.34), or, Step 1 of Procedure PA is repeated with a smaller integration step size and then Step 2 is solved via (3.36). We feel the latter to be better because, the failure of the first-order approach indicates a rapid change in g and so it is good to do a careful solution with a smaller integration step size. Our experience with numerical problems is that, cases where $\|X^*\| > \tau/2$ occurs are extremely rare.

The appropriate starting iterate for the MNR method applied to (3.37) is $X^0 = 0$, $\mu^0 = 0$. Let (x^l, μ^l) denote the lth iterate. It is easy to verify, using the structure of the system in (3.37), that the $(l+1)$th MNR iteration consists of:

(i) solving

$$G_X(0)G_X^t(0)\delta_\mu = -G(X^l) \tag{3.38}$$

for δ_μ; and

(ii) setting

$$\mu^{l+1} = \mu^l + \delta_\mu, \quad X^{l+1} = G_X^t(0)\mu^{l+1}. \tag{3.39}$$

Remark 3.10. It can be assumed that $\text{rank}(G_X(0)) = m$. This is reasonable because of Assumption 3.1 mentioned in Section 3.1 and the fact that $\bar{x}_{k+1}$ is close to $\mathcal{M}$. (Note that $G_X(0) = g_x(\bar{x}_{k+1})T$.)

An efficient, as well as accurate, way of solving (3.38) is by computing a skinny QR decomposition of $G_X^t(0)$, i.e., $G_X^t(0) = Q_1 R_1$, where $Q_1 \in \mathbb{R}^{n\times m}$ has orthonormal columns and $R_1 \in \mathbb{R}^{m\times m}$ is upper triangular [22]. Normally the QR decomposition of any matrix $B_{(n\times m)}$ $(n \geqslant m)$ is given by $B = QR$, with $R = (R_1, \mathbf{O})^t$. Q is an unitary matrix and R_1 is an upper triangular matrix. To avoid unnecessary computation, "skinny" QR decomposition is used [22], p. 217. The skinny QR decomposition of $A^t = G_X(0)^t = Q_1 R_1$ is unique, where $Q_1 \in \mathbb{R}^{n\times m}$ has orthonormal columns and R_1 is upper triangular with positive diagonal entries. To compute the above skinny QR decomposition, a modified Gram–Schmidt (MGS) orthogonalization process is used ([22], p. 219). This algorithm requires $2nm^2$ flops. The MGS computation is arranged so that A is overwritten by Q_1 and the matrix R_1 is stored in a separate array. Then (3.38) can be solved by replacing $G_X(0)G_X^t(0)$ by $R_1^t R_1$ and solving the resulting twin triangular systems.

Shampine [20] suggests a method that is equivalent to the Newton–Raphson method applied to (3.37). The first iterations of his method and our method are equivalent. Each additional iteration of his method requires the evaluation of a Jacobian of G and the computation of a partial QR decomposition of its transpose; whereas, our method does not require these computations.

Now consider the scaled l_∞-norm. Here the norm used in (3.25) is defined by $\|x\| = \|Wx\|_\infty$, where W is nonsingular. Usually W is diagonal. Define $T = W^{-1}$ and consider the transformation (3.33). Let G be as given in (3.35). Then (3.25) becomes

$$\min \ \|X\|_\infty \quad \text{s.t. } G(X) = 0. \tag{3.40}$$

The linearization of (3.40) around $X = 0$ is

$$\min \|X\|_\infty \quad \text{s.t. } AX = b, \tag{3.41}$$

where $A = G_X(0)$ and $b = -G(0)$. By Remark 3.10, it is reasonable to assume that $\text{rank}(A) = m$. The following lemma is useful for solving (3.41) and (3.40). It is the dual of 'Fundamental Theorem 2' of Cadzow [40].

Lemma 3.3. *There exists a solution, X of (3.41) with the property that at least $(n - m + 1)$ components of X are equal in absolute value and the columns of A corresponding to the remaining components are linearly independent.*

Proof. Problem (3.41) is equivalent to the LP problem,

$$\min w \quad \text{s.t. } AX = b; \ w \geqslant X^i, \ w \geqslant -X^i, \ i = 1, \ldots, n. \tag{3.42}$$

The solution is attained by an extreme point of the feasible set and, the property mentioned in the lemma is nothing but the characterization of such a point. □

Cadzow [40] gives an efficient algorithm for finding a solution of (3.41) with the property mentioned in Lemma 3.3. Basically, the algorithm is a careful specialization of the simplex method to (3.42). Initially, the algorithm chooses $(m-1)$ linearly independent columns of A, sets signs for the components of X corresponding to the remaining columns, assigns all these components to be equal in absolute value to a quantity w, and then solves the square system of equations resulting from $AX = b$ to obtain X. Then a simple test checks the optimality of X. If X is not optimal, an appropriate change is made to the choice of $(m-1)$ columns of A so that a descent in the cost is assured. These steps, which represent one iteration of the algorithm, are repeated until a solution is found. In the worst case, the number of iterations needed for convergence is exponential in m. Typically, however, convergence takes place in less than n iterations.

Now consider the solution of (3.40) for use in Procedure PA. Remark 3.9 extends to the scaled l_∞-norm. By Remark 3.5 it is sufficient to find an X^* satisfying

$$G(X) = 0 \quad \|X\|_\infty \leqslant \tau/2. \tag{3.43}$$

Verifying whether X^* satisfies (3.43) is much easier than checking if X^* solves (3.40). Following is a procedure for solving (3.43). Typically, the X^* returned by it also solves (3.40).

Procedure PC. Finding an X^* that satisfies (3.43).

1. Solve (3.41) by Cadzow's algorithm. Let X_0 be the solution found. Generically, $(n-m+1)$ components of X_0, with index list I, are equal in absolute value, say w_0, and the absolute values of the remaining components are strictly less than w_0. Let us assume this to be the case.
2. If $G(X) = 0$ (in a numerical implementation this is checked using a tolerance), set $\bar{X} = X_0$ and go to Step 3. If not, do the following. Choose a variable w and set: $X^i = \text{sign}(X_0^i)w \ \forall i \in I$. Let $z \in \mathbb{R}^m$ denote the vector of variables containing $\{X^i\colon\ i \notin I\} \cup \{w\}$. Then solve the square nonlinear system in z resulting from $G(X) = 0$, by the MNR method to obtain a solution, $\bar{z}$. Let $\bar{X}$ denote the corresponding solution in the X space.
3. If $\|\bar{X}\|_\infty \leqslant \tau/2$, stop with $X^* = \bar{X}$. Else, indicate failure and stop. (As mentioned in Remark 3.9, a good remedy for the failure is to go back to Step 1 of Procedure PA and decrease the integration step size.)

An important implementation detail is worth mentioning here. In each of its iterations Cadzow's algorithm maintains and updates the inverse of an $m \times m$ matrix, $\tilde{A}$ which has the structure that the first $(m-1)$ of its columns are

columns of A and the last column is formed by adding/subtracting the remaining columns of A. The inverse of $\tilde{A}$ available at the termination of Cadzow's algorithm, i.e., Step 1 of Procedure PC, serves as an excellent approximation of the inverse required by the MNR method in Step 2.

Following are two good reasons which suggest that the scaled l_∞-norm correction is cheaper than the scaled l_2-norm correction: (i) the former can be implemented using the LU factorization of an $m \times m$ matrix, whereas the latter requires the more expensive partial QR decomposition of an $n \times m$ matrix; and (ii) the index list I at the end of a call to Procedure PC serves well as the index list needed for initializing Cadzow's algorithm during the next call to Procedure PC, i.e., the next integration step. Even with a random choice for the index list I, Cadzow [40] has observed that solving (3.41) using his algorithm is usually cheaper than doing an iteration of the form (3.38), (3.39).

4. Comparison

In this section we compare the various approaches. The comparison is done both, by doing a crude cost analysis and by studying the performance on a set of numerical examples. Since the choice of an integration method is as important as the choice of the approach itself we first give a quick review of some well-established integration methods for solving ODEs.

4.1. Numerical solution of ODEs

There are various integration methods available for solving the ODE of the form (2.2). In many applications we come across two type of ODEs: (i) Stiff; and, (ii) Nonstiff based on the qualitative behavior of the ODE. If (2.2) is stiff an implicit multistep method [25] due to Gear is the best choice. If (2.2) is nonstiff an explicit method is appropriate. In particular, there are two popular methods: Runge–Kutta method and Adams. Several fine codes based on these above methods are available in DEPAC (Sandia), ODEPACK (Lawrence Livermore), NAG and IMSL libraries. Both methods produce approximants x_n and f_n, respectively to $x(t_n)$ and $\dot{x}(t_n)$. At the nth time step, x_{n+1} and f_{n+1} are generated by using the known approximants at previous mesh points. The step-size, $h_{n+1} = t_{n+1} - t_n$ is chosen as long as possible while still meeting an error criterion on the solution specified by the user.

Adams methods are multistep methods that make use of information at previous time points to obtain x_{n+1}. Of the Adams methods, Adams–Bashforth–Moulton Predictor Corrector (PECE) formulas is the most popularly used. Suppose we have approximation x_{n-1} to the solution $x(t_{n-i})$ for $i = 0, 1, \ldots, k$, where k is the order of the method used (up to 12 in most codes).

We would like to find an approximation to the solution at t_{n+1}. First, an initial guess for the solution at t_{n+1} is formed by evaluating the predictor polynomial and the derivative at t_{n+1} by evaluating the derivative function f. The approximation x_{n+1} to the solution at t_{n+1} which is finally accepted is the solution of the corrector formula. While each step costs only two f-evaluations, the overhead in each step (the tasks of forming the predictor polynomial, selecting and changing the order and step size etc.) is high. Because of these features, Adams methods are attractive for problems requiring high accuracy and having expensive f-evaluation. In the case of CMS each f-evaluation involves the solution of the linear system.

RK methods are single step methods which generate from $x_n, f(x_n)$ and several more f-evaluation in the interval $[t_n, t_{n+1}]$. RK methods of a fixed low order are efficient for problems needing a moderate accuracy and having cheap f-evaluation and RK method of 4th order is the most popular in this respect. Among various RK methods, one due to Fehlberg has enjoyed popularity. The method is based on a pair of formulas of orders four and five. In particular, the code RKF45 (renamed as DERKF and available in DEPAC) written by Shampine and Watts has been widely used. However, the development of a new pair of four–five formulas by Dormand and Prince [41] and its subsequent analysis and modifications by Shampine [42] have led to the new pair, DPS. The DPS pair is cleverly structured and the parameters in its formulas are systematically chosen to minimize the truncation errors. Shampine [42] has shown DPS is much better than the Fehlberg pair by all the key performance standards. Therefore DPS is bound to replace the Fehlberg pair in future RK codes. The code RKDPS written by Sudarsan and Keerthi [43] is available and has been tested and passed all the verifications. We have modified this code for CMS. The preliminary investigations have shown some promising results and a detailed study is needed (authors have not done a detailed study of RK methods for DAEs). Hairer et al. [7] studied extensively the application of RK methods to DAEs. In Ref. [44] convergence of RK methods for DAEs of index-3 is discussed.

Another important thing to consider in this context is interpolation. Interpolant is used to approximate the solution and its derivative between mesh points. This facility can be used for handling event functions and also for graphical outputs. The polynomial produced for the popular Fehlberg(4,5) Runge–Kutta formulas by Horn [45] and that in the Adams code of Shampine and Gordon [37] do not connect at the mesh points to form a globally continuous function. Those of the BDF and Adams code of Gear [46] form a continuous function, but the first derivative has jump discontinuities. The jumps seen in these codes are comparable in size to the local error tolerances. To remedy this difficulty, a number of authors have provided algorithms with C^1 piecewise polynomial approximations for Runge–Kutta [41,42], Adams [47], and BDF [48] codes.

4.2. Cost analysis

We now carry out a simple cost (of computation) analysis that reveals which choice of approach–integration method combination is the most efficient for a given problem [49]. This analysis is rather crude. It is only meant to give a rough quantification of the effect of various factors on cost. Thus the expressions for costs derived here should not be taken to match closely with costs associated with actual numerical solutions.

Let us define:

$\tilde{n}$ = the number of integrated variables
N_{steps} = number of integration steps needed to complete an integration task
C_{int} = cost (average) of doing one integration step
c_f = cost of evaluating one derivative function and
n_f = number of derivative function evaluations per step

If (2.2) is directly integrated, then $\tilde{n} = n$. N_{steps} depends on the problem solved, the method used and the way the method is implemented. It has been generally observed that RK45 takes much larger steps than Adams and that

$$[N_{\text{steps}}]_{\text{Adams}} = 2[N_{\text{steps}}]_{\text{RK45}} \tag{4.1}$$

is a good estimate. The C_{int} is given by [49]

$$C_{\text{int}} = \beta_0 + \beta_1 \tilde{n}, \tag{4.2}$$

where β_0 and β_1 are constants associated with the integration method and the way it is implemented. Using the data of Shampine et al. [49] obtained from detailed numerical testing it is possible to roughly estimate β_0 and β_1 for the RK45 and Adams codes of [4]. Such values are given in Table 2. Note that the values are much bigger for Adams. The value of n_f is

$$n_f = \begin{cases} 6 & \text{for RK45;} \\ 2 & \text{for Adams.} \end{cases} \tag{4.3}$$

Thus, if the aim is to integrate (2.2) directly,

$$\text{integration cost} = N_{\text{steps}}(C_{\text{int}} + n_f c_f). \tag{4.4}$$

Table 2
Constants associated with C_{int}

Integration code	β_0	β_1
RK45	85.71	40.0
Adams	292.06	62.86

β_0 and β_1 are in number of equivalent arithmetic operations.

By (4.1) and Table 2 the term $N_{\text{steps}}C_{\text{int}}$ is much bigger for Adams. However the remaining term, $N_{\text{steps}}n_f c_f$ is smaller for Adams. Thus, the choice of method for efficient integration depends on how big c_f is in comparison with C_{int}. A good rule of thumb is to use RK45 whenever the derivative function is defined by simple expressions and is cheap to compute. This is typically the case in control system simulation. When the derivative function is expensive to evaluate Adams method is more efficient. Such a case usually occurs when each derivative function evaluation involves the solution of a subproblem such as nonlinear equation solving; for example, the ODE in (3.9).

Before comparing the efficiency of the various approaches let us look at other factors of comparison. Interpolation for x is much easier to do with inexact constraint stabilization approach (ICS) and ECS than with others [50]. Consider next, the ease of implementation, i.e., the programming effort needed to modify an available ODE code so that it solves the vector field (2.1), (2.2). The ordering of the approaches with respect to increasing difficulty of implementation is: ICS, PA, ECS, TP and CP. ICS only requires the programming of (3.21) and (3.22). TP needs the implementation of Procedure TP and the steps for forming a parameterization and checking its validity. Both ICS and TP do not require any alteration of the ODE code. PA needs the implementation of Step 2 of Procedure PA and the proper insertion of this extra code at the end of an integration step of the ODE code. To improve efficiency, ECS has to incorporate the modifications of DASSL. CP is the most difficult to implement because the extrapolator for z has to be maintained in the same data structure as that used by the ODE code for storing the interpolant of y. This requires a thorough understanding of the ODE code.

For efficiency, the choice of approach depends on the problem being solved. Later we shall make specific recommendations for some of the applications mentioned already. We now extend the cost analysis which we did for the direct integration of (2.2) to the solution of (2.1), (2.2) by an approach in combination with an integration method. The analysis is somewhat crude, but observations made using it correspond well with those made from numerical testing.

If (2.2) is stiff, then ECS, which solves the stabilized index two form of Euler–Lagrange equation using an appropriate modification of the DAE code DASSL, is the best choice. The solution of CMS with stiffness using CP and TP is addressed in Refs. [51,52].

Consider the case where (2.2) is nonstiff. Here ECS is quite inefficient compared to others. Suppose one of the approaches other than ECS is used in combination with one of the integration methods, RK45 and Adams. Let C_{step} denote the cost (average) of one time step of the solution process. It is useful to classify C_{step} as

$$C_{\text{step}} = C_{\text{int}} + C_{\text{mc}} + C_{\text{eval}}, \tag{4.5}$$

where C_{int} is the cost (average) associated with the integration method as defined earlier; C_{mc} the cost (average) required in doing any manifold-correction computations, e.g., Step 1 of Procedure TP, Step 2 of Procedure PA, and the computation of the term $g_x^{\text{t}}(x)K(x)$ in (3.22) via (3.21); and C_{eval} the cost of evaluating the basic functions f, g and g_x in one step.

Let us consider these subcosts one by one. C_{int} is given by (4.2) where

$$\tilde{n} = \begin{cases} (n-m) & \text{for TP and CP;} \\ n & \text{for others.} \end{cases} \tag{4.6}$$

Actually, when a multistep integration method, such as Adams is used, $\tilde{n} = n$ should be used for CP because it maintains an extrapolator for the dependent variable set, z, which is nearly as expensive as integrating z.

Next consider the component C_{mc}. To avoid a complicated analysis, let us make the reasonable assumption that on the average, manifold-correction involves the formation and factorization of an $m \times m$ matrix and one solution of a twin triangular linear system of equations (e.g., the operations involved up to a single iteration of the MNR method applied to implement Step 1 of Procedure TP). Therefore a good approximation for C_{mc} is

$$C_{\text{mc}} = \begin{cases} \text{O}(nm^2) & \text{for PA;} \\ n_f \text{O}(nm^2) & \text{for others.} \end{cases} \tag{4.7}$$

Here $\text{O}(nm^2)$ denotes a polynomial in n and m whose highest order term is nm^2. Though it is slightly different for the various approaches, we can take them to be equal for our analysis here. In fact, for the case of n and m being large, $\text{O}(nm^2) = nm^2$ is a good estimate, where cost is measured in terms of the number of equivalent arithmetic operations. Using the assumption on manifold-correction made here, we can also obtain

$$C_{\text{eval}} = n_f c_f + \begin{cases} (2c_g + c_{gx}) & \text{for PA;} \\ n_f(2c_g + c_{gx}) & \text{for others,} \end{cases} \tag{4.8}$$

where c_f, c_g and c_{gx} are, respectively, the costs of evaluating f, g and gx.

Let N_{steps} denote the number of time steps needed to solve a given problem. Given an integration method and a tolerance for it, it is reasonable to assume that all approaches will require about the same number of steps. Since PA works with half the tolerance as others, we can use Remark 3.7 and further assume

$$[N_{\text{steps}}]_{\text{PA}} = 1.15N, \tag{4.9}$$

where N is the value of N_{steps} for the other approaches.

Putting (4.5)–(4.9) together and using (4.2) we get C, the cost of solving (2.1), (2.2) as

$$C = \begin{cases} N[\beta_0 + \beta_1(n-m) + n_f \mathrm{O}(nm^2) + n_f c_f + n_f(2c_g + c_{gx})] \\ \quad \text{for TP and CP;} \\ 1.15N[\beta_0 + \beta_1 n + \mathrm{O}(nm^2) + n_f c_f + (2c_g + c_{gx})] \\ \quad \text{for PA;} \\ N[\beta_0 + \beta_1 n + n_f \mathrm{O}(nm^2) + n_f c_f + n_f(2c_g + c_{gx})] \\ \quad \text{for ICS.} \end{cases} \tag{4.10}$$

Thus C is an interplay of several factors. If we take the effect of C_{eval} on C to be roughly the same for all approaches and so compare only the costs, we can make the following observations.

1. TP and CP are more efficient than ICS and ECS; the difference in their costs however, is only $\beta_1 m$ per step.
2. When the ratio m/n is bounded below by some positive constant and n is large, then the dominating term $1.15N\mathrm{O}(nm^2)$ in the cost of PA is smaller than the dominating term, $Nn_f\mathrm{O}(nm^2)$ in the cost of the other approaches. In such a case, PA is much more efficient than the others. This is especially true when the RK45 method is used for integration.
3. For small n and m, TP and CP have a slight edge over PA because they integrate only $(n-m)$ variables. For a specific problem, a user can make a reasonable decision on the choice of an efficient approach by: taking $\mathrm{O}(nm^2) = nm^2$ (in number of multiplies); using Table 2; and perhaps including estimates of c_f, c_g and c_{gx} for his problem in (4.10).

Let us now make specific recommendations for some of the applications mentioned. In control systems simulation, an f-evaluation is usually cheap. So it is best to use RK45 for integration. PA is a good approach for use with RK45. For CMS the situation is opposite since each f-evaluation requires $\mathrm{O}(n^3)$ effort and so it is expensive. Adams is the appropriate choice for integration. For small n, m, TP and CP are suitable. However, for large n, m (in most practical CMS n is large and the ratio m/n is 0.3 or even bigger) PA should be preferred. Similar choices hold well for homotopy curve tracing. Since accurate curve tracing is unimportant for homotopy methods, large tolerances should be used [53]. For the numerical curve tracing problem arising in geometric modeling, efficiency is not a key factor because solutions are usually done during the pre-processing stage. What is more important is to minimize N_{steps}. N_{steps} denotes the number of 'curve patches' that form the approximate curve of intersection, and, these are involved in on-line computations such as checking the intersection of the curve with another surface. By (4.1), RK45 is the best choice. By (4.9), other approaches are slightly better than PA.

4.3. Numerical examples

Clearly, it is not possible to single out a particular approach as the most suitable for solving any vector field. The choice of an appropriate approach-

integration method combination for a given application has to be made by looking at special structures in the vector field, by doing the cost analysis of Section 4.2, and, very importantly, by studying the performance of various combinations on selected numerical examples. Here we only consider vector fields associated with CMS. The Euler–Lagrange equation for CMS is given in the form

$$M(q)\ddot{q} + J^{t}(q)\lambda = Q(\dot{q}, q), \tag{4.11}$$

$$\phi(q) = 0. \tag{4.12}$$

The four approaches discussed in this paper are implemented in FORTRAN. A computer code called DAMES (Dynamic Analysis of MEchanical Systems) was developed for this purpose. The code DAMES can handle the equations of the form (4.11), (4.12). The inputs to DAMES are (1) the generalized Mass matrix $M(q)$ (full or diagonal, a flag is provided for this purpose to handle them efficiently), (2) the function $\phi(q)$ specifying the constraints, (3) the generalized external forces $Q(\dot{q}, q)$, (4) the Jacobian matrix J and, (5) the function $v(\dot{q}, q)$.

The Jacobian matrix J can either be provided by the user or may ask the code to compute the numerical Jacobian. In the case of ECS the function v need not be specified as it does not involve $\ddot{\phi}$. A carefully considered and explicitly stated experimental design is crucial in making valid inferences about the performance of the mathematical software. Developing a sound experimental design involves identifying the variables expected to be influential in determining the code performance, deciding the appropriate measures of performance. Choosing the appropriate performance indicators is a crucial factor in computational experiments. We have chosen performance indicators which are as independent as possible of the problem at hand. The following performance indicators are common to all the approaches: CPU–CPU Time (calculated in Micro-Vax – under ULTRIX-32m version 1.2 OS); NF-Number of Function evaluations (f-evaluations), NS-Number of integration steps to complete the integration from t_0 to t_f, NJ-Number of J matrix evaluation, NI-Average number of iterations taken to solve the nonlinear square system; and Nϕ-Number of ϕ evaluations. Apart from these, the indicators which are specific to particular approach are: NFK-Number of LU factorizations (Step 1 of Procedure TP), NT-Number of triangular systems solved (Step 1 of Procedure TP, solving for μ in the PA, Step 3 of Procedure ECS), NP-Number of new TP done. Before discussing the examples let us make the following remark.

Remark 4.1. As already mentioned, the selection of the stabilizing parameter in the case of ICS approach is a crucial issue and no clear way of choosing it is available. So in all the examples the parameter is chosen in the following way. We start with $D = 0$ (i.e., integrating just the underlying ODE) and slowly

increase its value. By looking at the solution plots a proper (optimal) D is chosen. So while actually comparing the performance indicators of this approach with the other approaches it is very important to keep in mind the exercise involved in choosing this optimal D.

In the case of TP approach, choosing the ratio μ is very important. Number of new parameterizations done depends mainly on this optimal choice of μ. Also as already mentioned the number of new parameterizations done also depends on ρ, the rate of convergence.

Example 4.1 (*Simple pendulum*). Consider the simple pendulum with the generalized coordinate $q = (x, y)^t$.The following are the inputs to DAMES. $M(q) = \text{diag}(m, m)$, $Q(\dot{q}, q) = (0, -mg)^t$, $\phi(q) = x^2 + y^2 - 1$, $J(q) = (2x\ 2y)$, $v(\dot{q}, q) = -(2x\dot{x}^2 + 2y\dot{y}^2)$, $[t_0, t_f] = [0, 2]$ s, $m = 1$ Kg, $g = 9.81$ m/s^2, $x = 1$ m, $y = 0$ m, $u = 0$ m/s, $v = 5$ m/s

Remarks: The consistent initial condition (CIC) is such that the pendulum is moved to the horizontal position and is thrown upwards with 5 m/s so that the pendulum completes two full circles. This system is simulated using the four approaches PA, TP, ECS, ICS and the results are tabulated in Table 3.

Example 4.2 (*Slider-crank mechanism*). Consider the elementary model of a slider-crank mechanism shown in Fig. 1 where the units of length are in meters. The crank (body-1) rotates without friction about an axis perpendicular to the x–y plane. The connecting rod (body-2) is constrained so that its center point slides without friction along the x-axis. The polar moment of inertia of bodies 1 and 2 are J_1 and J_2 kg m^2, respectively, and the mass of body-2 is m_2 (kg). A constant torque $T = 10$ N m is applied to body-1. The generalized coordinate vector $q = (\phi_1, x_2, \phi_2)^t$. Let us now define the inputs to DAMES.

$$M(q) = \text{diag}(J_1, m_2, J_2) \qquad J(q) = \begin{bmatrix} -\sin\phi_1 & -1 & -2\sin\phi_2 \\ \cos\phi_1 & 0 & 2\cos\phi_2 \end{bmatrix},$$

$$\phi(q) = \begin{bmatrix} \cos\phi_1 + 2\cos\phi_2 - x_2 \\ \sin\phi_1 + 2\sin\phi_2 \end{bmatrix},$$

$$Q(\dot{q}, q) = (T, 0, 0)^t, \quad v(\dot{q}, q) = \begin{bmatrix} \cos\phi_1\, \dot{\phi}_1^2 + 2\cos\phi_2\, \dot{\phi}_2^2 \\ \sin\phi_1\, \dot{\phi}_1^2 + 2\sin\phi_2\, \dot{\phi}_2^2 \end{bmatrix}.$$

Time interval $[t_0, t_f] = [0, 2]$ s $J_1 = 1$, $J_2 = 2$, $m_2 = 2$. The CIC are $\phi_1 = \pi/4$ rad, $x_2 = 2.5779$ m, $\phi_2 = -0.3613$ rad and zero initial velocity. This system

Table 3
Example: Simple pendulum

	PA	TP	ECS	ICS
TOL $=10^{-6}$				
NF	228	314	–	183
NS	114	157	473	92
NJ	233	727	528	188
Nϕ	192	810	1356	188
CPU	3.9	5.0	6.3	1.7
NI	1.6	1.8	1.99	–
NT	610	1169	2912	–
NFK	–	359	–	–
NP	–	8	–	–
TOL $=10^{-4}$				
NF	132	255	–	113
NS	61	128	287	74
NJ	137	653	358	118
Nϕ	120	682	728	118
CPU	2.5	4.4	4.8	1.3
NI	1.82	2.2	1.99	–
NT	370	1001	1276	–
NFK	–	320	–	–
NP	–	12	–	–
TOL $=10^{-2}$				
NF	70	82	–	60
NS	35	41	135	31
NJ	75	410	173	65
Nϕ	64	402	360	65
CPU	1.4	1.9	2.0	0.8
NI	1.82	1.92	1.98	–
NT	196	610	990	–
NFK	–	242	–	–
NP	–	16	–	–

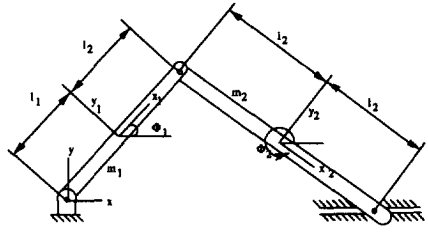

Fig. 1. Elementary slider-crank mechanism.

Table 4
Slider-crank mechanism

	PA	TP	ECS	ICS
TOL $=10^{-6}$				
NF	286	200	–	266
NS	145	100	580	135
NJ	291	455	468	271
Nϕ	233	468	3372	271
CPU	5.7	6.3	7.9	3.8
NI	1.61	2.1	2.2	–
NT	748	693	4817	–
NFK	–	225	–	–
NP	–	4	–	–
TOL $=10^{-4}$				
NF	162	121	–	166
NS	83	61	398	83
NJ	167	275	386	171
Nϕ	126	287	2340	171
CPU	3.0	3.9	5.1	1.5
NI	1.51	1.97	2.2	–
NT	410	423	3100	–
NFK	–	136	–	–
NP	–	2	–	–
TOL $=10^{-2}$				
NF	68	55	–	68
NS	36	28	186	36
NJ	73	143	172	73
Nϕ	63	165	988	73
CPU	1.8	1.9	3.1	1.0
NI	1.75	1.97	2.3	–
NT	190	234	1686	–
NFK	–	70	–	–
NP	–	2	–	–

is simulated using the four approaches PA, TP, ECS, ICS and the results are tabulated in Table 4.

Example 4.3 (*Three link cylindrical coordinate manipulator*). In this a three link coordinate cylindrical manipulator shown in Fig. 2 is considered. We assume that the joints of the robot are all rigid.

The end of the robot is constrained to move in one-dimensional path, namely a circle in the x–z plane with y being a constant. The constraint function in Cartesian coordinates is given by $(y-0.169=0,\ x^2+z^2-d=0)^t$, $d=0.221423$. The generalized coordinate vector $q=(q_1, q_2, q_3)^t$ as shown in the Fig. 2. The inputs are: $M(q)=\mathrm{diag}(J_1+J_2+J_3+m_3(q_3+l_3)^2,\ m_2+m_3,\ m_3)$

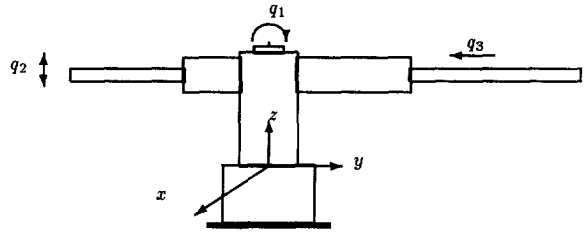

Fig. 2. Three link cylindrical coordinate manipulator.

$$Q(\dot{q},q)=\begin{bmatrix}2m_3(q_3+l_3)\dot{q}_1\dot{q}_3\\(m_2+m_3)g\\-m_3(q_3+l_3)\dot{q}_1^2\end{bmatrix},$$

$$J(q)=\begin{bmatrix}(q_3+l_3)\cos q_1 & 0 & \sin q_1\\-2(q_3+l_3)^2\cos q_1\sin q_1 & 2q_2 & 2(q_3+l_3)\cos^2 q_1\end{bmatrix},$$

$$\phi(q)=\begin{bmatrix}(q_3+l_3)\sin q_1-0.169\\(q_3+l_3)^2\cos^2 q_1+q_2^2-d\end{bmatrix},$$

$$v(\dot{q},q)=\begin{bmatrix}(q_3+l_3)\sin q_1\,\dot{q}_1^2-2\cos q_1\,\dot{q}_1\dot{q}_3\\2(q_3+l_3)^2\cos 2q_1\,\dot{q}_1^2-2\dot{q}_2^2-2\cos^2 q_1\,\dot{q}_3^2+\\4(q_3+l_3)\sin 2q_1\,\dot{q}_1\dot{q}_3\end{bmatrix}.$$

Time interval $[t_0,t_f]=[0,4]$ s, $m_2=1$ kg, $m_3=2$ kg, $J_1=0.1$ kg m^2, $J_2=0.2$ kg m^2, $J_3=0.1$ kg m^2, $l_3=0.2$ m CIC $q_1=0.48607$, $q_2=0.34512$, $q_3=0.16176$ and zero initial velocity. The control law for stabilizing the system to the equilibrium point, $q_e=(0.436,\ 0.3,\ 0.2)^t$, designed using a linearized approach [54] is given by

$$\begin{aligned}u=&0.363-3.655(q_1-0.436)-4.181(q_2-0.3)+3.136(q_3-0.2)\\&-0.455\dot{q}_1-2.167\dot{q}_2+1.083\dot{q}_3\\&29.4-15.938(q_1-0.436)-18.229(q_2-0.3)+13.672(q_3-0.2)\\&-2.167\dot{q}_1-10.325\dot{q}_2+5.163\dot{q}_3\\&0.423+8.267(q_1-0.436)+9.455(q_2-0.3)-7.091(q_3-0.2)\\&+1.083\dot{q}_1+5.163\dot{q}_2-2.581\dot{q}_3\end{aligned}$$

This system is simulated using the four approaches PA, TP, ECS, ICS and the results are tabulated in Table 5.

Example 4.4 (*Quick-return mechanism*). As an example of the many compound mechanisms that arise in practice, the quick-return mechanism of Fig. 3 that

Table 5
Three link cylindrical coordinate manipulator

	PA	TP	ECS	ICS
$TOL = 10^{-6}$				
NF	302	378	–	298
NS	151	183	340	150
NJ	308	1034	328	303
Nϕ	260	1030	1680	303
CPU	6.1	7.2	9.1	3.1
NI	1.72	3.1	2.2	–
NT	822	1370	2218	–
NFK	–	390	–	–
NP	–	4	–	–
$TOL = 10^{-4}$				
NF	190	210	–	178
NS	95	105	256	90
NJ	195	460	240	183
Nϕ	155	472	1396	183
CPU	3.8	4.5	6.2	1.9
NI	1.63	2.1	2.5	–
NT	498	698	1548	–
NFK	–	235	–	–
NP	–	6	–	–
$TOL = 10^{-2}$				
NF	85	110	–	88
NS	43	55	105	45
NJ	91	245	98	93
Nϕ	79	220	512	93
CPU	2.0	2.5	3.8	1.1
NI	1.8	1.8	2.0	–
NT	242	394	624	–
NFK	–	130	–	–
NP	–	8	–	–

represents a shaper is considered. With counterclockwise rotation of the crank (body-3), cutting occurs as the tool (body-6) moves to the left through the workpiece. The quick-return stroke of the tool occurs as it moves to the right. In the model of Fig. 3, each link is modeled as a body. The elements of the model are as follows: Body-1 is ground, and the body-fixed frames are as shown in the figure and the constraints are defined in Table 6. The generalized coordinate vector $q = (x_i, y_i, z_i)^t$ for $i = 1, \ldots, 6$.

$$M(q) = \mathrm{diag}(m_i,\ m_i,\ m_i I_i^3) \quad \text{for } i = 1, \ldots, 6,$$
$$Q(\dot{q}, q) = (0, -m_1 g, 0, 0, -m_2 g, 0, 0, -m_3 g, T_3, 0, -m_4 g, 0, 0, -m_5 g, 0, 0, -m_6 g, 0)^t,$$

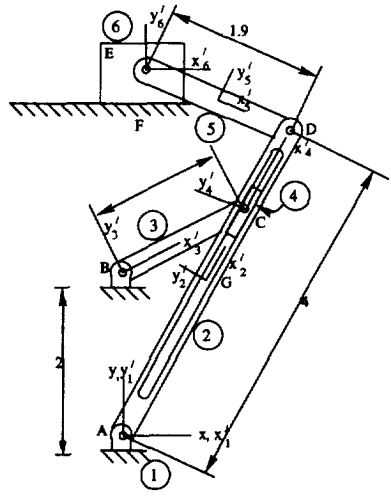

Fig. 3. Quick-return mechanism.

Table 6
Inertial properties and CIC

No.	1	2	3	4	5	6
Mass	1	100	1000	5.0	30	60
MI	1	100	2000	0.5	10	1.5
x	0	0.91822	0	1.35993	0.91296	−0.01053
y	0	1.77676	2	2.63293	3.77637	3.99923
ϕ	0	1.90380	0.4356	1.90380	0.23680	0

$$\begin{aligned}\phi(q)\ (&x_1, y_1, \phi_1, x_1 - x_2 + 2\cos\phi_2, y_1 - y_2 + 2\sin\phi_2, x_1 - x_3 - 2\sin\phi_1,\\ &y_1 - y_3 + 2\cos\phi_1, x_3 - x_4 + 1.5\cos\phi_3, y_3 - y_4 + 1.5\sin\phi_3.\\ &x_5 - x_2 + 0.95\cos\phi_5 - 2\cos\phi_2, y_5 - y_2 + 0.95\sin\phi_5 - 2\sin\phi_2,\\ &x_5 - x_6 - 0.95\cos\phi_5,\\ &y_5 - y_6 - 0.95\sin\phi_5, (x_4 - x_2)\sin\phi_2 - (y_4 - y_2)\cos\phi_2,\\ &\sin(\phi_4 - \phi_2), (x_6 - x_1)\sin\phi_1 - (y_6 - y_1)\cos\phi_1, \sin(\phi_6 - \phi_1))^t.\end{aligned}$$

The nonzero elements of the Jacobian matrix $J(q)$ and $v(\dot{q}, q)$ can be computed easily.

Remarks: Dynamic analysis is carried out with a slider mass $m_6 = 50$ kg, a torque $T_3 = 165,521$ N m applied to flywheel, and flywheel polar moment of inertia (MI) is 200 kg m^2 and with initial velocity zero. The applied torque is selected so that the work done in one cycle of operation ($2\pi T_3$) is equal to

Table 7
Quick-return mechanism

	PA	TP	ECS	ICS
$TOL = 10^{-6}$				
NF	2160	3112	–	2082
NS	1080	1556	3574	1044
NJ	2060	9512	3787	2087
Nϕ	2054	9482	11232	2087
CPU	77.2	86.3	98.5	52.2
NI	1.98	2.5	2.3	–
NT	6190	12012	18612	–
NFK	–	4312	–	–
NP	–	14	–	–
$TOL = 10^{-4}$				
NF	1272	1876	–	1191
NS	686	938	1932	599
NJ	1173	5724	1860	1196
Nϕ	1166	5430	8632	1196
CPU	45.5	58.9	78.7	29.9
NI	1.98	1.99	2.3	–
NT	3524	9876	12712	–
NFK	–	2842	–	–
NP	–	16	–	–
$TOL = 10^{-2}$				
NF	712	1012	–	676
NS	356	506	1208	346
NJ	682	3814	1194	681
Nϕ	745	3712	5977	681
CPU	24.3	34.8	48.2	17.1
NI	1.9	2.0	2.1	–
NT	1930	6812	8308	–
NFK	–	1864	–	–
NP	–	10	–	–

the work done in cutting the workpiece. This system is simulated using the four approaches PA, TP, ECS, ICS and the results are tabulated in Table 7.

4.4. Observations and recommendations

From the tables it appears that ICS is the most efficient approach. But, as already remarked (Remark 4.1) the selection of the stabilizing parameter $D = \alpha$ in the case of ICS approach is a crucial issue and no clear general way is available. In the examples we have tried, the parameter α is chosen in the following way. We start with $\alpha = 0$ (i.e., integrating just the underlying ODE) and slowly increase its value. By looking at the amount of constraint

violation and the number of failed integration steps a proper (optimal) α is chosen. So while actually comparing the performance indicators of this approach with the other approaches it is very important to keep in mind the huge exercise involved in choosing this optimal α. Since the selection of α is highly problem dependent the ICS is not a suitable approach for general purpose use. Therefore in our comparison of the different approaches we exclude it.

Based on the data from the numerical examples we make the following observations. (1) In terms of all the performance indicators, PA is the best, in spite of the 1.15 factor in Eq. (4.10). (2) ECS takes substantially more integration steps for all the examples as expected. (3) NJ, Nϕ, and NT are substantially more for TP and ECS and so the effort is more. (4) Finally in the case of TP, the parameter NFK (Step 1 of Procedure TP_CMS) reflects the extra effort in the case of TP and is of the order NS. Also, choosing the ratio μ is very important. The number of new parameterizations done depends mainly on this optimal choice of μ. Also as repeatedly mentioned the number of new parameterizations done also depends on ρ, the rate of convergence.

Finally we recommend the following. To start with, solve the given CMS using ICS approach with some $\alpha \neq 0$. If the integration fails repeatedly to accept a step and the stepsize becomes too small and the minimum stepsize is reached, and also if the constraint violation is too large, abort this approach. Now start the more accurate and involved approach PA and use it with stiffness detection in the integration routine. The ideas in Refs. [55,56] can be used for this purpose. The code DAMES detects stiffness using these ideas. If stiffness is detected, abort PA and use ECS.

5. Conclusion

The PA proposed in Ref. [50] overcomes some of the difficulties faced by the other existing approaches. The salient features of this approach are: (i) it decouples the process of integration from the process of correction to the constraint manifold; (ii) it solves the vector fields in terms of the original coordinates and hence the integration tolerances for the original variables can be specified directly; and (iii) it does not involve any coordinate transformation which has to be carried out in some of the other approaches and hence there is no necessity for integration restart.

An important issue concerning integration error control has not been carefully addressed in the literature for Parameterization approach. In this paper we suggest a way for specifying these tolerances which is very important in any good implementation of the Parameterization approach.

We argue that the PA is better than the Parameterization approach. The chief defects of the Parameterization approach are that: (i) each f evaluation requires the solution of an m-dimensional nonlinear system of equations; and

(ii) since the parameterization is local, a change in parameterization may be required during the solution, leading to an integration restart with associated inefficiencies. The PA does not suffer from these defects. It requires only one solution of an m-dimensional nonlinear system of equations in each integration step (Step 2 of Procedure PA). Also, it does not require any integration restarts because it deals with the full ODE system in (2.2). The Parameterization approach has the advantage that it integrates only the $(n - m)$-dimensional system of ODEs, whereas the PA requires the integration of the n-dimensional system of ODEs. This advantage, however, is only slight because the difference in the integration overhead costs of the two approaches is only $O(m)$ whereas the cost of every extra m-dimensional nonlinear system solution required by the Parameterization approach is $O(m^3)$. The four approaches (PA, TP, ECS, ICS) have been carefully coded and compared on some test problems. The PA has a number of advantages over existing approaches. The qualitative comparisons are mentioned above.

References

[1] P. Kunkel, V. Mehrmann, Canonical forms for linear differential-algebraic equations with variable coefficients, JCAM 56 (1994) 225–251.
[2] P.J. Rabier, W.C. Rheinboldt, Classical and generalized solutions of time-dependent linear differential-algebraic equations, Linear Algebra Applications 245 (1996) 259–293.
[3] P.J. Rabier, W.C. Rheinboldt, A general existence and uniqueness theorem for implicit differential-algebraic equations, Technical Report ICMA-90-145, Dept. of Mathematics and Statistics, University of Pittsburgh, 1990.
[4] L.F. Shampine, H.A. Watts, DEPAC – Design of user oriented package of ODE solvers, SAND-79-2374, SNL, Albuquerque, 1980.
[5] L.F. Shampine, H.A. Watts, Software for ordinary differential equations, in: W.R. Cowell (Ed.), Sources and Development of Mathematical Software, Prentice-Hall, Englewood Cliffs, NJ, 1984.
[6] E. Hairer, S.P. Norsett, G. Wanner, Solving ordinary differential equations, Non-stiff problems, 2nd ed., Springer, Berlin, 1993.
[7] E. Hairer, G. Wanner, Stiff and Differential Algebraic Problems, Springer, Berlin, 1991.
[8] M. Roche, Runge Kutta methods for differential algebraic equations: Theory and implementation, ZAMM 69 (4) (1989) T35–T36.
[9] R.D. Skeel, Thirteen ways to estimate global error, Num. Math. 48 (1986) 1–20.
[10] H. Ashrafioun, R. Colbert, L. Obergefell, I. Kaleps, Modeling of a deformable manikin neck for multibody dynamic simulation, Math. Comput. Modeling 24 (2) (1996) 45–56.
[11] W.S. Yoo, E.J. Haug, Dynamics of flexible mechanical systems using vibration and static correction modes, J. Mech. Trans. Auto. Design 108 (1986) 315–322.
[12] U. Ascher, On numerical differential algebraic problems with applications to semiconductor device simulation, SIAM J. Num. Anal. 26 (3) (1989) 517–538.
[13] K. Venugopal, S.S. Keerthi, On the approximate parametrization of the curve of intersections of implicit surfaces, Appl. Math. Lett. 1 (1991) 100–111.
[14] F.E. Udwadia, R.E. Kalaba, An alternate proof for the equation of motion for constrained mechanical systems, Appl. Math. Comp. 70 (1995) 339–342.

[15] R. Barrett et al., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, Philadelphia, PA, 1993.
[16] W. Hackbusch, Iterative Solution of Large Sparse Systems of Equations, Springer, Berlin, 1994.
[17] K.E. Brenan, S.L. Campbell, L.R. Petzold, Numerical Solution of Initial-value Problems in Differential–Algebraic Equations, 2nd ed., Elsevier, Amsterdam, 1989.
[18] L.R. Petzold, A description of DASSL: A differential/algebraic system solver, in: R.S. Stepleman et al. (Ed.), Scientific Computing, North-Holland, Amsterdam, 1983, pp. 65–68.
[19] C.C. Pantelides, The consistent initialization of differential–algebraic systems, SIAM J. Sci. Stat. Comp. 9 (2) (1988) 213–231.
[20] L.F. Shampine, Conservation laws and the numerical solution of ODEs, Computers Math. Appl. 12B (5/6) (1986) 1287–1296.
[21] J.M. Ortega, W.C. Rheinboldt, Iterative methods for nonlinear equations in several variables, Academic Press, New York 1970.
[22] G.H. Golub, C.F. Van Loan, Matrix Computations, 2nd ed., Johns Hopkins University Press, Baltimore, MD, 1989.
[23] D. Goldberg, What every computer scientist should know about floating-point arithmetic, ACM Computing Surveys 23 (1991) 5–48.
[24] W.J. Cody, ALGORITHM 665–MACHAR: A subroutine to dynamically determine machine parameters, ACM Trans. Math. Soft. 14 (4) (1988) 303–311.
[25] L.F. Shampine, Implementation of implicit formulas for the solution of ODEs, SIAM J. Sci. Stat. Comp. 1 (1980) 103–118.
[26] R.A. Wehage, E.J. Haug, Generalized coordinate partitioning for dimension reduction in analysis of constrained dynamic systems, Trans. ASME J. Mech. Design 104 (1982) 247–255.
[27] N.K. Mani, E.J. Haug, K.E. Atkinson, Application of singular value decomposition for analysis of mechanical system dynamics, Trans. ASME J. Mech. Trans. Aut. Design 107 (1985) 82–87.
[28] E.J. Haug, Computer Aided Kinematics and Dynamics of Mechanical Systems: Basic Methods, vol. I, Allyn & Bacon, Boston, 1989.
[29] T.W. Park, E.J. Haug, A hybrid numerical integration method for machine dynamic simulation, Trans, ASME J. Mech. Trans. Aut. Design 108 (1986) 211–216.
[30] C.W. Gear, Maintaining solution invariants in the numerical solution of ODEs, SIAM J. Sci. Stat. Comp. 7 (3) (1986) 734–743.
[31] J. Baumgarte, Stabilization of constraints and integrals of motion in dynamical systems, Comp. Meth. Appl. Mech. Engg. 1 (1972) 1–16.
[32] E. Eich, M. Hanke, Regularization methods for constrained mechanical multibody systems, ZAMM 75 (10) (1995) 761–773.
[33] C.W. Gear, Differential–algebraic equations index transformations, SIAM J. Sci. Stat. Comp. 9 (1) (1988) 39–47.
[34] C. Tischendorf, Feasibility and stability behaviour of the BDF applied to index-2 differential algebraic equations, ZAMM 75 (12) (1995) 927–946.
[35] J. Xiang, I.T. Cameron, Numerical solution of DAE systems using block BDF methods, J. Comput. Appl. Math. 62 (3) (1995) 255–266.
[36] F.T. Krogh, Changing stepsize in the numerical integration of differential equations using modified differences, in: Proc. Conf. Num. Solution of ODEs, Lecture Notes in Math., No. 362, Springer, New York, 1974.
[37] L.F. Shampine, M.K. Gordon, Computer Solution of Ordinary Differential Equations, Freeman, New York, 1975.
[38] K.R. Jackson, R. Sacks-Davis, An alternative implementation of variable stepsize multistep formulas for stiff ODEs, ACM Trans. Math. Soft. 6 (1980) 295–318.

[39] R. Sudarsan, S.S. Keerthi, An efficient approach for the simulation of multibody system dynamics, 1993, communicated.
[40] J.A. Cadzow, A finite algorithm for the minimum l_∞ solution to system of consistent linear equations, SIAM J. Num. Anal. 10 (1973) 607–617.
[41] J.R. Dormand, P.J. Prince, Runge–Kutta triples, Comput. Math. Appls. 12 (1986) 1007–1017.
[42] L.F. Shampine, Some practical RK formulas, Math. Comp. 46 (1986) 135–150.
[43] R. Sudarsan, S.S. Keerthi, Description of the code RKDPS – Dormand–Prince–Shampine RK formulas, Technical Report CS-90, Indian Institute of Science, Bangalore, 1990.
[44] L. Jay, Convergence of Runge–Kutta methods for differential-algebraic systems of index 3, Appl. Numer. Math. 17 (2) (1995) 97–118.
[45] M.K. Horn, Fourth and fifth-order scaled Runge–Kutta algorithms for treating dense output, SIAM J. Num. Anal. 20 (1983) 558–568.
[46] C.W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations, Prentice-Hall, Englewood Cliffs, NJ, 1971.
[47] H.A. Watts, L.F. Shampine, Smoother interpolants for Adams codes, SIAM J. Sci. Stat. Comp. 7 (1986) 334–345.
[48] M.A. Berzins, A C^1 interpolant for codes based on backward difference formulas, Numer. Math. 2 (1986) 109–118.
[49] L.F. Shampine, H.A. Watts, S.M. Davenport, Solving non stiff ordinary differential equations – The state of the art, SIAM Rev. 18 (1976) 376–411.
[50] R. Sudarsan, A new approach for the numerical solution of constrained mechanical systems, Ph.D. Thesis, Indian Institute of Science, Bangalore, India, 1992.
[51] E.J. Haug, J. Yen, Implicit numerical integration of constrained equations via generalized coordinate partitioning, Technical Report R-39, Center of Simulation and Design, University of Iowa, 1989.
[52] F.A. Potra, J. Yen, Implicit numerical integration for Euler–Lagrange equations via tangent space parametrization, Mechanics Structures Machines 19 (1991) 77–98.
[53] W.C. Rheinboldt, Numerical Analysis of Parametrized Nonlinear Equations, Wiley, New York, 1986.
[54] H. Krishnan, N.H. McClamroch, A new approach to position and contact free regulation in constrained robot systems, in: Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati 1990.
[55] L.F. Shampine, Stiffness and nonstiff differential equation solversII: Detecting stiffness with Runge–Kutta methods, ACM Trans. Math. Soft. 3 (1977) 44–53.
[56] L.F. Shampine, K.L. Hiebert, Detecting stiffness with Fehlberg (4,5) formulas, Comp. Math. Appls. 3 (1977) 41–46.